ITMO UNIVERSITY

How to Win Coding Competitions: Secrets of Champions

Week 4: Algorithms on Graphs
Lecture 2: Graphs: Representations in memory

Maxim Buzdalov
Saint Petersburg 2016

Two main ways to store a graph in computer memory are:

- Adjacency matrix
- Adjacency list

Two main ways to store a graph in computer memory are:

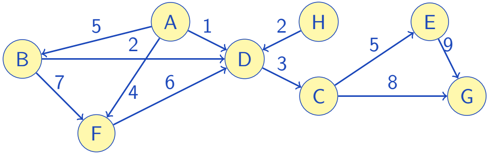
- Adjacency matrix
- Adjacency list

These ways are different in the following aspects:

- Space complexity (expressed in $|V|$, $|E|$)
- Running time of various operations
  - Vertex insertion
  - Edge insertion, edge deletion
  - Edge existence test
  - Iteration over edges adjacent to a vertex

The graph $G = (V, E)$ without multiedges with weight function F is represented as the matrix A of size $|V| \times |V|$ in the following manner. For each ordered pair of vertices u and v with $(u, v) \in E$, the matrix stores $A[u][v] = F((u, v))$.
All other cells of A are filled by a neutral value (typically zero).

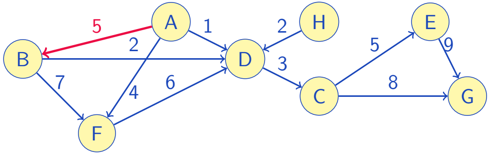
The graph $G = (V, E)$ without multiedges with weight function F is represented as the matrix A of size $|V| \times |V|$ in the following manner. For each ordered pair of vertices u and v with $(u, v) \in E$, the matrix stores $A[u][v] = F((u, v))$.
All other cells of A are filled by a neutral value (typically zero).



	A	B	C	D	E	F	G	H
A	–	5	–	1	–	2	–	–
B	–	–	–	2	–	7	–	–
C	–	–	–	–	5	–	8	–
D	–	–	3	–	–	–	–	–
E	–	–	–	–	–	–	9	–
F	–	–	–	6	–	–	–	–
G	–	–	–	–	–	–	–	–
H	–	–	–	2	–	–	–	–

- Space – $\Theta(|V|^2)$
- Vertex insertion – $\Theta(|V|)$
- Edge insertion, deletion, testing – $\Theta(1)$
- Adjacent edge iteration – $\Theta(n)$

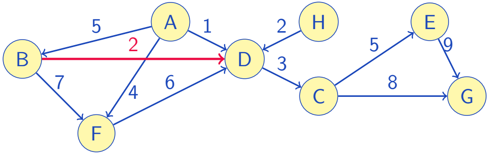
The graph $G = (V, E)$ without multiedges with weight function F is represented as the matrix A of size $|V| \times |V|$ in the following manner. For each ordered pair of vertices u and v with $(u, v) \in E$, the matrix stores $A[u][v] = F((u, v))$.
All other cells of A are filled by a neutral value (typically zero).



	A	B	C	D	E	F	G	H
A	–	5	–	1	–	2	–	–
B	–	–	–	2	–	7	–	–
C	–	–	–	–	5	–	8	–
D	–	–	3	–	–	–	–	–
E	–	–	–	–	–	–	9	–
F	–	–	–	6	–	–	–	–
G	–	–	–	–	–	–	–	–
H	–	–	–	2	–	–	–	–

- Space – $\Theta(|V|^2)$
- Vertex insertion – $\Theta(|V|)$
- Edge insertion, deletion, testing – $\Theta(1)$
- Adjacent edge iteration – $\Theta(n)$

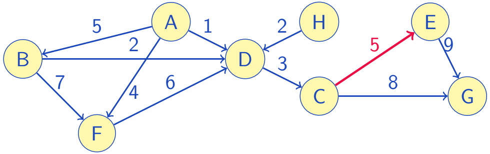
The graph $G = (V, E)$ without multiedges with weight function F is represented as the matrix A of size $|V| \times |V|$ in the following manner. For each ordered pair of vertices u and v with $(u, v) \in E$, the matrix stores $A[u][v] = F((u, v))$.
All other cells of A are filled by a neutral value (typically zero).



	A	B	C	D	E	F	G	H
A	–	5	–	1	–	2	–	–
B	–	–	–	2	–	7	–	–
C	–	–	–	–	5	–	8	–
D	–	–	3	–	–	–	–	–
E	–	–	–	–	–	–	9	–
F	–	–	–	6	–	–	–	–
G	–	–	–	–	–	–	–	–
H	–	–	–	2	–	–	–	–

- Space – $\Theta(|V|^2)$
- Vertex insertion – $\Theta(|V|)$
- Edge insertion, deletion, testing – $\Theta(1)$
- Adjacent edge iteration – $\Theta(n)$

The graph $G = (V, E)$ without multiedges with weight function F is represented as the matrix A of size $|V| \times |V|$ in the following manner. For each ordered pair of vertices u and v with $(u, v) \in E$, the matrix stores $A[u][v] = F((u, v))$.
All other cells of A are filled by a neutral value (typically zero).



	A	B	C	D	E	F	G	H
A	–	5	–	1	–	2	–	–
B	–	–	–	2	–	7	–	–
C	–	–	–	–	5	–	8	–
D	–	–	3	–	–	–	–	–
E	–	–	–	–	–	–	9	–
F	–	–	–	6	–	–	–	–
G	–	–	–	–	–	–	–	–
H	–	–	–	2	–	–	–	–

- Space – $\Theta(|V|^2)$
- Vertex insertion – $\Theta(|V|)$
- Edge insertion, deletion, testing – $\Theta(1)$
- Adjacent edge iteration – $\Theta(n)$

Example: Check if there exists a cycle of length 3 in the given undirected graph

Example: Check if there exists a cycle of length 3 in the given undirected graph

A simple straightforward algorithm:

```
function TriangleExistence(A)
    n ← Rows(A)
    for u from 1 to n do
        for v from u + 1 to n do
            if A[u][v] = 1 then continue end if
            for w from v + 1 to n do
                if A[u][w] = 1 and A[v][w] = 1 then return TRUE end if
            end for
        end for
    end for
end function
```

Example: Check if there exists a cycle of length 3 in the given undirected graph

A simple straightforward algorithm:

```
function TriangleExistence(A)
    n ← Rows(A)
    for u from 1 to n do                                              ▷ Checking all u
        for v from u + 1 to n do
            if A[u][v] = 1 then continue end if
            for w from v + 1 to n do
                if A[u][w] = 1 and A[v][w] = 1 then return true end if
            end for
        end for
    end for
end function
```

Example: Check if there exists a cycle of length 3 in the given undirected graph

A simple straightforward algorithm:

```
function TriangleExistence(A)
    n ← Rows(A)
    for u from 1 to n do                                          ▷ Checking all u
        for v from u + 1 to n do                                  ▷ Checking all v
            if A[u][v] = 1 then continue end if
            for w from v + 1 to n do
                if A[u][w] = 1 and A[v][w] = 1 then return TRUE end if
            end for
        end for
    end for
end function
```

Example: Check if there exists a cycle of length 3 in the given undirected graph

A simple straightforward algorithm:

```
function TriangleExistence(A)
    n ← Rows(A)
    for u from 1 to n do                                          ▷ Checking all u
        for v from u + 1 to n do                                  ▷ Checking all v
            if A[u][v] = 1 then continue end if
            for w from v + 1 to n do                              ▷ Checking all w
                if A[u][w] = 1 and A[v][w] = 1 then return TRUE end if
            end for
        end for
    end for
end function
```

Example: Check if there exists a cycle of length 3 in the given undirected graph

A simple straightforward algorithm:

```
function TriangleExistence(A)
    n ← Rows(A)
    for u from 1 to n do                                        ▷ Checking all u
        for v from u + 1 to n do                                ▷ Checking all v
            if A[u][v] = 1 then continue end if
            for w from v + 1 to n do                            ▷ Checking all w
                if A[u][w] = 1 and A[v][w] = 1 then return TRUE end if
            end for
        end for
    end for
end function
```

Running time: $O(|V|^3)$.

Example: Check if there exists a cycle of length 3 in the given undirected graph

A simple straightforward algorithm:

```
function TriangleExistence(A)
    n ← Rows(A)
    for u from 1 to n do                                          ▷ Checking all u
        for v from u + 1 to n do                                  ▷ Checking all v
            if A[u][v] = 1 then continue end if
            for w from v + 1 to n do                              ▷ Checking all w
                if A[u][w] = 1 and A[v][w] = 1 then return TRUE end if
            end for
        end for
    end for
end function
```

Running time: $O(|V|^3)$. Can we make it faster?

Improvement idea: Do things “in parallel” using bitwise operations!

Improvement idea: Do things "in parallel" using bitwise operations!
Compressed matrix: store $A[i][j]$ as bits of 32 or 64-bit integers (example: 8 bits)

Improvement idea: Do things “in parallel” using bitwise operations!
Compressed matrix: store $A[i][j]$ as bits of 32 or 64-bit integers (example: 8 bits)

0	1	0	1	1	1	0	1	0	0	1	1	0	0	1	0
0	0	1	1	0	1	0	1	0	1	1	0	1	0	1	0
0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	1
1	1	0	1	1	0	0	1	1	0	1	1	1	1	1	0
1	1	1	0	0	1	1	1	1	0	1	1	1	1	0	0
0	1	1	0	0	1	0	0	0	0	1	1	0	1	1	1
1	1	1	1	1	1	1	0	1	1	0	0	1	0	0	1
1	1	0	1	0	1	0	0	0	1	1	0	0	1	0	0
0	1	0	1	1	0	1	1	1	1	1	0	0	0	1	1
1	0	0	0	0	1	1	0	1	0	0	0	1	0	1	0
0	0	1	0	0	1	1	0	1	1	1	0	0	0	1	0
1	1	0	0	1	1	0	1	0	0	1	0	0	0	0	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1
1	1	1	1	0	0	1	0	0	0	1	0	1	1	0	0
1	0	0	1	1	0	1	1	1	0	0	1	0	0	0	1
0	1	0	1	0	1	1	1	0	0	0	0	1	1	0	0

186	76
172	86
112	131
155	125
231	61
38	236
127	147
43	38
218	199
97	81
100	71
179	132
254	159
79	52
217	137
234	48

Improvement idea: Do things “in parallel” using bitwise operations!

Compressed matrix: store $A[i][j]$ as bits of 32 or 64-bit integers (example: 8 bits)

0	1	0	1	1	1	0	1	0	0	1	1	0	0	1	0
0	0	1	1	0	1	0	1	0	1	1	0	1	0	1	0
0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	1
1	1	0	1	1	0	0	1	1	0	1	1	1	1	1	0
1	1	1	0	0	1	1	1	1	0	1	1	1	1	0	0
0	1	1	0	0	1	0	0	0	0	1	1	0	1	1	1
1	1	1	1	1	1	1	0	1	1	0	0	1	0	0	1
1	1	0	1	0	1	0	0	0	1	1	0	0	1	0	0
0	1	0	1	1	0	1	1	1	1	1	0	0	0	1	1
1	0	0	0	0	1	1	0	1	0	0	0	1	0	1	0
0	0	1	0	0	1	1	0	1	1	1	0	0	0	1	0
1	1	0	0	1	1	0	1	0	0	1	0	0	0	0	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1
1	1	1	1	0	0	1	0	0	0	1	0	1	1	0	0
1	0	0	1	1	0	1	1	1	0	0	1	0	0	0	1
0	1	0	1	0	1	1	1	0	0	0	0	1	1	0	0

186	76
172	86
112	131
155	125
231	61
38	236
127	147
43	38
218	199
97	81
100	71
179	132
254	159
79	52
217	137
234	48

Improvement idea: Do things “in parallel” using bitwise operations!

Compressed matrix: store $A[i][j]$ as bits of 32 or 64-bit integers (example: 8 bits)

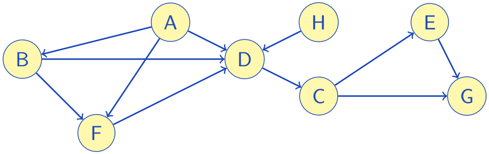
0	1	0	1	1	1	0	1	0	0	1	1	0	0	1	0
0	0	1	1	0	1	0	1	0	1	1	0	1	0	1	0
0	0	0	0	1	1	1	0	1	1	0	0	0	0	0	1
1	1	0	1	1	0	0	1	1	0	1	1	1	1	1	0
1	1	1	0	0	1	1	1	1	0	1	1	1	1	0	0
0	1	1	0	0	1	0	0	0	0	1	1	0	1	1	1
1	1	1	1	1	1	1	0	1	1	0	0	1	0	0	1
1	1	0	1	0	1	0	0	0	1	1	0	0	1	0	0
0	1	0	1	1	0	1	1	1	1	1	0	0	0	1	1
1	0	0	0	0	1	1	0	1	0	0	0	1	0	1	0
0	0	1	0	0	1	1	0	1	1	1	0	0	0	1	0
1	1	0	0	1	1	0	1	0	0	1	0	0	0	0	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1
1	1	1	1	0	0	1	0	0	0	1	0	1	1	0	0
1	0	0	1	1	0	1	1	1	0	0	1	0	0	0	1
0	1	0	1	0	1	1	1	0	0	0	0	1	1	0	0

186	76
172	86
112	131
155	125
231	61
38	236
127	147
43	38
218	199
97	81
100	71
179	132
254	159
79	52
217	137
234	48

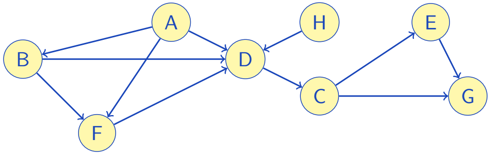
A (slightly simplified) bitmask-optimized version which works 32 times faster!

```
function TriangleExistence(A)
    n ← Rows(A)
    C ← BitmaskCompress(A)
    for u from 1 to n do
        for v from u + 1 to n do
            if A[u][v] = 1 then continue end if
            for w from (v + 1)/32 to (n + 31)/32 do
                if (C[u][w] bitwise and C[v][w]) ≠ 0 then return TRUE end if
            end for
        end for
    end for
end function
```

Given a graph G, find the number of paths of length k.

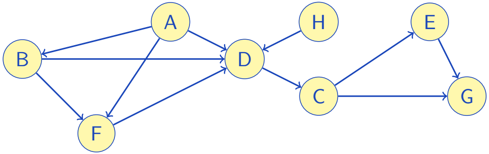


Given a graph G, find the number of paths of length k.



- Hint 1: Adjacency matrix = paths of length 1

Given a graph G, find the number of paths of length k.

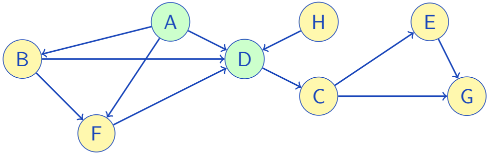


- Hint 1: Adjacency matrix = paths of length 1

$k = 1$

	A	B	C	D	E	F	G	H
A	0	1	0	1	0	1	0	0
B	0	0	0	1	0	1	0	0
C	0	0	0	0	1	0	1	0
D	0	0	1	0	0	0	0	0
E	0	0	0	0	0	0	1	0
F	0	0	0	1	0	0	0	0
G	0	0	0	0	0	0	0	0
H	0	0	0	1	0	0	0	0

Given a graph G, find the number of paths of length k.

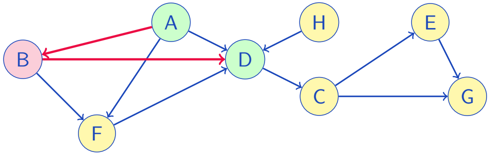


$k = 1$

	A	B	C	D	E	F	G	H
A	0	1	0	1	0	1	0	0
B	0	0	0	1	0	1	0	0
C	0	0	0	0	1	0	1	0
D	0	0	1	0	0	0	0	0
E	0	0	0	0	0	0	1	0
F	0	0	0	1	0	0	0	0
G	0	0	0	0	0	0	0	0
H	0	0	0	1	0	0	0	0

- Hint 1: Adjacency matrix = paths of length 1
- Hint 2: What is 2-path between A and D?

Given a graph G, find the number of paths of length k.

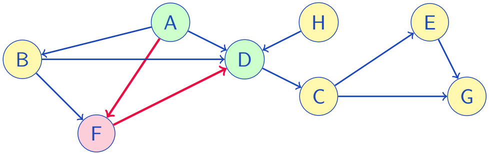


$k = 1$

	A	B	C	D	E	F	G	H
A	0	1	0	1	0	1	0	0
B	0	0	0	1	0	1	0	0
C	0	0	0	0	1	0	1	0
D	0	0	1	0	0	0	0	0
E	0	0	0	0	0	0	1	0
F	0	0	0	1	0	0	0	0
G	0	0	0	0	0	0	0	0
H	0	0	0	1	0	0	0	0

- Hint 1: Adjacency matrix = paths of length 1
- Hint 2: What is 2-path between A and D?

Given a graph G, find the number of paths of length k.

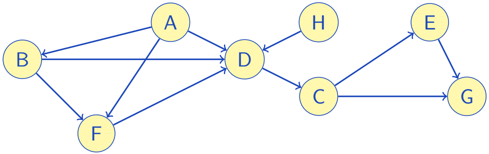


- Hint 1: Adjacency matrix = paths of length 1
- Hint 2: What is 2-path between A and D?

$k = 1$

	A	B	C	D	E	F	G	H
A	0	1	0	1	0	1	0	0
B	0	0	0	1	0	1	0	0
C	0	0	0	0	1	0	1	0
D	0	0	1	0	0	0	0	0
E	0	0	0	0	0	0	1	0
F	0	0	0	1	0	0	0	0
G	0	0	0	0	0	0	0	0
H	0	0	0	1	0	0	0	0

Given a graph G, find the number of paths of length k.



- Hint 1: Adjacency matrix = paths of length 1
- Hint 2: What is 2-path between A and D?

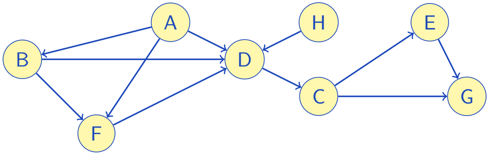
$k = 1$

	A	B	C	D	E	F	G	H
A	0	1	0	1	0	1	0	0
B	0	0	0	1	0	1	0	0
C	0	0	0	0	1	0	1	0
D	0	0	1	0	0	0	0	0
E	0	0	0	0	0	0	1	0
F	0	0	0	1	0	0	0	0
G	0	0	0	0	0	0	0	0
H	0	0	0	1	0	0	0	0

$k = 2$

	A	B	C	D	E	F	G	H
A	?	?	?	2	?	?	?	?
B	?	?	?	?	?	?	?	?
C	?	?	?	?	?	?	?	?
D	?	?	?	?	?	?	?	?
E	?	?	?	?	?	?	?	?
F	?	?	?	?	?	?	?	?
G	?	?	?	?	?	?	?	?
H	?	?	?	?	?	?	?	?

Given a graph G, find the number of paths of length k.



- Hint 1: Adjacency matrix = paths of length 1
- Hint 2: What is 2-path between A and D?
- Hint 3: $A_2[i][j] = \sum_k A_1[i][k] \cdot A_1[k][j]$

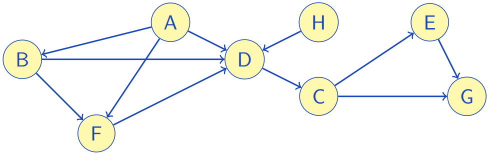
$k = 1$

	A	B	C	D	E	F	G	H
A	0	1	0	1	0	1	0	0
B	0	0	0	1	0	1	0	0
C	0	0	0	0	1	0	1	0
D	0	0	1	0	0	0	0	0
E	0	0	0	0	0	0	1	0
F	0	0	0	1	0	0	0	0
G	0	0	0	0	0	0	0	0
H	0	0	0	1	0	0	0	0

$k = 2$

	A	B	C	D	E	F	G	H
A	?	?	?	2	?	?	?	?
B	?	?	?	?	?	?	?	?
C	?	?	?	?	?	?	?	?
D	?	?	?	?	?	?	?	?
E	?	?	?	?	?	?	?	?
F	?	?	?	?	?	?	?	?
G	?	?	?	?	?	?	?	?
H	?	?	?	?	?	?	?	?

Given a graph G, find the number of paths of length k.



- Hint 1: Adjacency matrix = paths of length 1
- Hint 2: What is 2-path between A and D?
- Hint 3: $A_2[i][j] = \sum_k A_1[i][k] \cdot A_1[k][j]$
  - or simply $A_2 = A_1 \cdot A_1 = (A_1)^2$

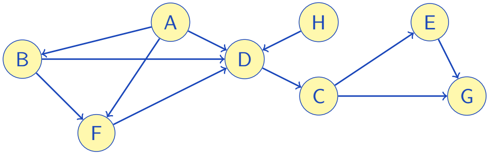
$k = 1$

	A	B	C	D	E	F	G	H
A	0	1	0	1	0	1	0	0
B	0	0	0	1	0	1	0	0
C	0	0	0	0	1	0	1	0
D	0	0	1	0	0	0	0	0
E	0	0	0	0	0	0	1	0
F	0	0	0	1	0	0	0	0
G	0	0	0	0	0	0	0	0
H	0	0	0	1	0	0	0	0

$k = 2$

	A	B	C	D	E	F	G	H
A	?	?	?	2	?	?	?	?
B	?	?	?	?	?	?	?	?
C	?	?	?	?	?	?	?	?
D	?	?	?	?	?	?	?	?
E	?	?	?	?	?	?	?	?
F	?	?	?	?	?	?	?	?
G	?	?	?	?	?	?	?	?
H	?	?	?	?	?	?	?	?

Given a graph G, find the number of paths of length k.



- Hint 1: Adjacency matrix = paths of length 1
- Hint 2: What is 2-path between A and D?
- Hint 3: $A_2[i][j] = \sum_k A_1[i][k] \cdot A_1[k][j]$
  - or simply $A_2 = A_1 \cdot A_1 = (A_1)^2$

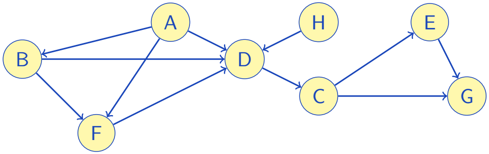
$k = 1$

	A	B	C	D	E	F	G	H
A	0	1	0	1	0	1	0	0
B	0	0	0	1	0	1	0	0
C	0	0	0	0	1	0	1	0
D	0	0	1	0	0	0	0	0
E	0	0	0	0	0	0	1	0
F	0	0	0	1	0	0	0	0
G	0	0	0	0	0	0	0	0
H	0	0	0	1	0	0	0	0

$k = 2$

	A	B	C	D	E	F	G	H
A	0	0	1	2	0	1	0	0
B	0	0	1	1	0	0	0	0
C	0	0	0	0	0	0	1	0
D	0	0	0	0	1	0	1	0
E	0	0	0	0	0	0	0	0
F	0	0	1	0	0	0	0	0
G	0	0	0	0	0	0	0	0
H	0	0	1	0	0	0	0	0

Given a graph G, find the number of paths of length k.



- Hint 1: Adjacency matrix = paths of length 1
- Hint 2: What is 2-path between A and D?
- Hint 3: $A_2[i][j] = \sum_k A_1[i][k] \cdot A_1[k][j]$
  - or simply $A_2 = A_1 \cdot A_1 = (A_1)^2$
- $A_k = (A_1)^k$, can be evaluated in $O(|V|^3 \log k)$
  - $O(|V|^3)$ (or faster): matrix multiplication

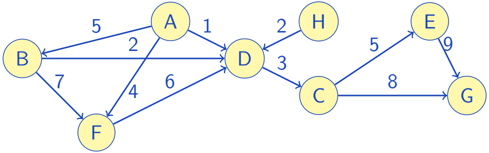
$k = 1$

	A	B	C	D	E	F	G	H
A	0	1	0	1	0	1	0	0
B	0	0	0	1	0	1	0	0
C	0	0	0	0	1	0	1	0
D	0	0	1	0	0	0	0	0
E	0	0	0	0	0	0	1	0
F	0	0	0	1	0	0	0	0
G	0	0	0	0	0	0	0	0
H	0	0	0	1	0	0	0	0

$k = 2$

	A	B	C	D	E	F	G	H
A	0	0	1	2	0	1	0	0
B	0	0	1	1	0	0	0	0
C	0	0	0	0	0	0	1	0
D	0	0	0	0	1	0	1	0
E	0	0	0	0	0	0	0	0
F	0	0	1	0	0	0	0	0
G	0	0	0	0	0	0	0	0
H	0	0	1	0	0	0	0	0

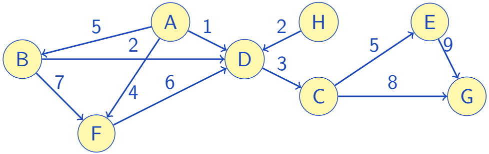
A compact storage for sparse graphs.
For every vertex, store outgoing edges.



	A	B	C	D	E	F	G	H
A	–	5	–	1	–	2	–	–
B	–	–	–	2	–	7	–	–
C	–	–	–	–	5	–	8	–
D	–	–	3	–	–	–	–	–
E	–	–	–	–	–	–	9	–
F	–	–	–	6	–	–	–	–
G	–	–	–	–	–	–	–	–
H	–	–	–	2	–	–	–	–

Adjacency list

A compact storage for sparse graphs.
For every vertex, store outgoing edges.

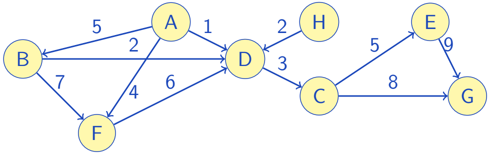


	A	B	C	D	E	F	G	H
A	–	5	–	1	–	2	–	–
B	–	–	–	2	–	7	–	–
C	–	–	–	–	5	–	8	–
D	–	–	3	–	–	–	–	–
E	–	–	–	–	–	–	9	–
F	–	–	–	6	–	–	–	–
G	–	–	–	–	–	–	–	–
H	–	–	–	2	–	–	–	–

A	$(B;5)$	$(D;1)$	$(F;2)$
B	$(D;2)$	$(F;7)$	
C	$(E;5)$	$(G;8)$	
D	$(C;3)$		
E	$(G;9)$		
F	$(D;6)$		
G			
H	$(D;2)$		

A compact storage for sparse graphs.

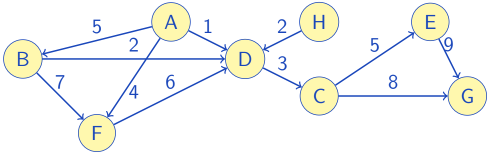
For every vertex, store incoming and outgoing edges.



	A	B	C	D	E	F	G	H
A	–	5	–	1	–	2	–	–
B	–	–	–	2	–	7	–	–
C	–	–	–	–	5	–	8	–
D	–	–	3	–	–	–	–	–
E	–	–	–	–	–	–	9	–
F	–	–	–	6	–	–	–	–
G	–	–	–	–	–	–	–	–
H	–	–	–	2	–	–	–	–

A	$(B;5)$	$(D;1)$	$(F;2)$			A
B	$(D;2)$	$(F;7)$			$(A;5)$	B
C	$(E;5)$	$(G;8)$			$(D;3)$	C
D	$(C;3)$	$(H;2)$	$(F;6)$	$(B;2)$	$(A;1)$	D
E	$(G;9)$				$(C;5)$	E
F	$(D;6)$			$(B;7)$	$(A;4)$	F
G				$(E;9)$	$(C;8)$	G
H	$(D;2)$					H

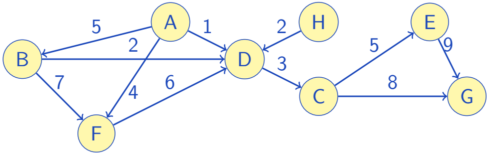
A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.



	A	B	C	D	E	F	G	H
A	–	5	–	1	–	2	–	–
B	–	–	–	2	–	7	–	–
C	–	–	–	–	5	–	8	–
D	–	–	3	–	–	–	–	–
E	–	–	–	–	–	–	9	–
F	–	–	–	6	–	–	–	–
G	–	–	–	–	–	–	–	–
H	–	–	–	2	–	–	–	–

A	$(B;5)$	$(D;1)$	$(F;2)$			A
B	$(D;2)$	$(F;7)$			$(A;5)$	B
C	$(E;5)$	$(G;8)$			$(D;3)$	C
D	$(C;3)$	$(H;2)$	$(F;6)$	$(B;2)$	$(A;1)$	D
E	$(G;9)$				$(C;5)$	E
F	$(D;6)$			$(B;7)$	$(A;4)$	F
G				$(E;9)$	$(C;8)$	G
H	$(D;2)$					H

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.

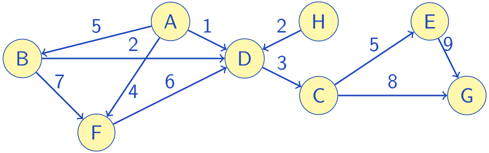


	A	B	C	D	E	F	G	H
A	–	5	–	1	–	2	–	–
B	–	–	–	2	–	7	–	–
C	–	–	–	–	5	–	8	–
D	–	–	3	–	–	–	–	–
E	–	–	–	–	–	–	9	–
F	–	–	–	6	–	–	–	–
G	–	–	–	–	–	–	–	–
H	–	–	–	2	–	–	–	–

A	$(B;5)$	$(D;1)$	$(F;2)$			A
B	$(D;2)$	$(F;7)$			$(A;5)$	B
C	$(E;5)$	$(G;8)$			$(D;3)$	C
D	$(C;3)$	$(H;2)$	$(F;6)$	$(B;2)$	$(A;1)$	D
E	$(G;9)$				$(C;5)$	E
F	$(D;6)$			$(B;7)$	$(A;4)$	F
G				$(E;9)$	$(C;8)$	G
H	$(D;2)$					H

- Space requirements: $\Theta(|V| + |E|)$

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.

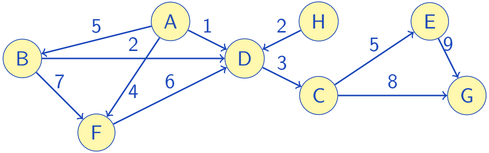


	A	B	C	D	E	F	G	H
A	–	5	–	1	–	2	–	–
B	–	–	–	2	–	7	–	–
C	–	–	–	–	5	–	8	–
D	–	–	3	–	–	–	–	–
E	–	–	–	–	–	–	9	–
F	–	–	–	6	–	–	–	–
G	–	–	–	–	–	–	–	–
H	–	–	–	2	–	–	–	–

A	$(B;5)$	$(D;1)$	$(F;2)$			A
B	$(D;2)$	$(F;7)$			$(A;5)$	B
C	$(E;5)$	$(G;8)$			$(D;3)$	C
D	$(C;3)$	$(H;2)$	$(F;6)$	$(B;2)$	$(A;1)$	D
E	$(G;9)$				$(C;5)$	E
F	$(D;6)$			$(B;7)$	$(A;4)$	F
G				$(E;9)$	$(C;8)$	G
H	$(D;2)$					H

- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.

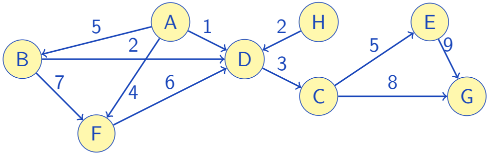


	A	B	C	D	E	F	G	H
A	–	5	–	1	–	2	–	–
B	–	–	–	2	–	7	–	–
C	–	–	–	–	5	–	8	–
D	–	–	3	–	–	–	–	–
E	–	–	–	–	–	–	9	–
F	–	–	–	6	–	–	–	–
G	–	–	–	–	–	–	–	–
H	–	–	–	2	–	–	–	–

A	$(B;5)$	$(D;1)$	$(F;2)$			A
B	$(D;2)$	$(F;7)$			$(A;5)$	B
C	$(E;5)$	$(G;8)$			$(D;3)$	C
D	$(C;3)$	$(H;2)$	$(F;6)$	$(B;2)$	$(A;1)$	D
E	$(G;9)$				$(C;5)$	E
F	$(D;6)$			$(B;7)$	$(A;4)$	F
G				$(E;9)$	$(C;8)$	G
H	$(D;2)$					H

- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.

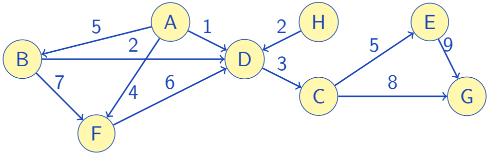


	A	B	C	D	E	F	G	H
A	–	5	–	1	–	2	–	–
B	–	–	–	2	–	7	–	–
C	–	–	–	–	5	–	8	–
D	–	–	3	–	–	–	–	–
E	–	–	–	–	–	–	9	–
F	–	–	–	6	–	–	–	–
G	–	–	–	–	–	–	–	–
H	–	–	–	2	–	–	–	–

A	$(B;5)$	$(D;1)$	$(F;2)$			A
B	$(D;2)$	$(F;7)$			$(A;5)$	B
C	$(E;5)$	$(G;8)$			$(D;3)$	C
D	$(C;3)$	$(H;2)$	$(F;6)$	$(B;2)$	$(A;1)$	D
E	$(G;9)$				$(C;5)$	E
F	$(D;6)$			$(B;7)$	$(A;4)$	F
G				$(E;9)$	$(C;8)$	G
H	$(D;2)$					H

- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.

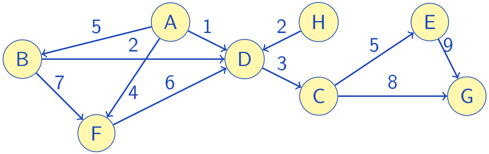


	A	B	C	D	E	F	G	H
A	–	5	–	1	–	2	–	–
B	–	–	–	2	–	7	–	–
C	–	–	–	–	5	–	8	–
D	–	–	3	–	–	–	–	–
E	–	–	–	–	–	–	9	–
F	–	–	–	6	–	–	–	–
G	–	–	–	–	–	–	–	–
H	–	–	–	2	–	–	–	–

A	$(B;5)$	$(D;1)$	$(F;2)$			A
B	$(D;2)$	$(F;7)$			$(A;5)$	B
C	$(E;5)$	$(G;8)$			$(D;3)$	C
D	$(C;3)$	$(H;2)$	$(F;6)$	$(B;2)$	$(A;1)$	D
E	$(G;9)$				$(C;5)$	E
F	$(D;6)$			$(B;7)$	$(A;4)$	F
G				$(E;9)$	$(C;8)$	G
H	$(D;2)$					H

- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$
  - $O(\log(deg(v)))$ if balanced search trees are used

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

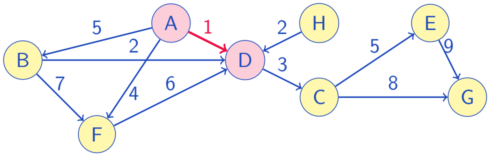


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	–	–	–	–	–	–	–	–

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	–	–	–	–	–	–	–	–	–	–	–
Value	–	–	–	–	–	–	–	–	–	–	–
Next	–	–	–	–	–	–	–	–	–	–	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant’s way: $O(1)$ dynamic data structures (outgoing only edges shown)

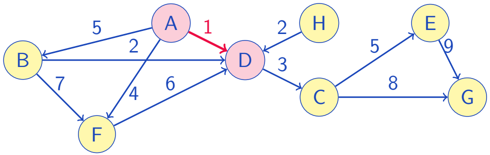


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	–	–	–	–	–	–	–	–

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	–	–	–	–	–	–	–	–	–	–	–
Value	–	–	–	–	–	–	–	–	–	–	–
Next	–	–	–	–	–	–	–	–	–	–	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

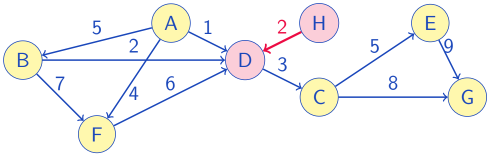


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	1	–	–	–	–	–	–	–

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	–	–	–	–	–	–	–	–	–	–
Value	1	–	–	–	–	–	–	–	–	–	–
Next	–	–	–	–	–	–	–	–	–	–	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

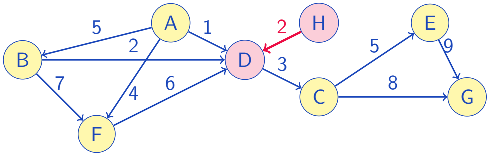


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	1	–	–	–	–	–	–	–

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	–	–	–	–	–	–	–	–	–	–
Value	1	–	–	–	–	–	–	–	–	–	–
Next	–	–	–	–	–	–	–	–	–	–	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)



- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

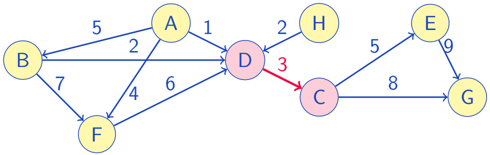
Vertex	A	B	C	D	E	F	G	H
Next	1	–	–	–	–	–	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	–	–	–	–	–	–	–	–	–
Value	1	2	–	–	–	–	–	–	–	–	–
Next	–	–	–	–	–	–	–	–	–	–	–

A compact storage for sparse graphs.

For every vertex, store incoming and outgoing edges.

The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

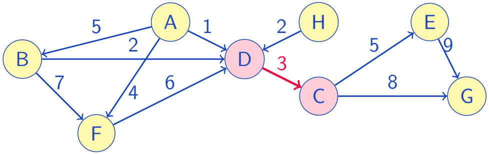


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	1	–	–	–	–	–	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	–	–	–	–	–	–	–	–	–
Value	1	2	–	–	–	–	–	–	–	–	–
Next	–	–	–	–	–	–	–	–	–	–	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)



- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

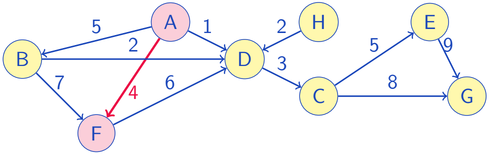
Vertex	A	B	C	D	E	F	G	H
Next	1	–	–	3	–	–	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	–	–	–	–	–	–	–	–
Value	1	2	3	–	–	–	–	–	–	–	–
Next	–	–	–	–	–	–	–	–	–	–	–

A compact storage for sparse graphs.

For every vertex, store incoming and outgoing edges.

The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

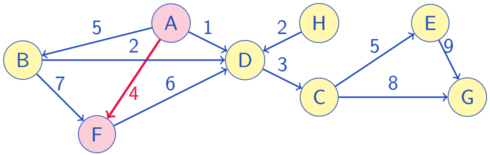


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	1	–	–	3	–	–	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	–	–	–	–	–	–	–	–
Value	1	2	3	–	–	–	–	–	–	–	–
Next	–	–	–	–	–	–	–	–	–	–	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

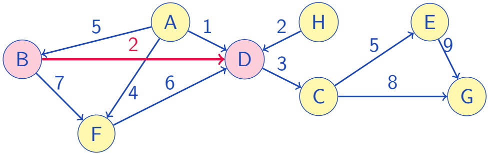


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	4	–	–	3	–	–	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	–	–	–	–	–	–	–
Value	1	2	3	4	–	–	–	–	–	–	–
Next	–	–	–	1	–	–	–	–	–	–	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

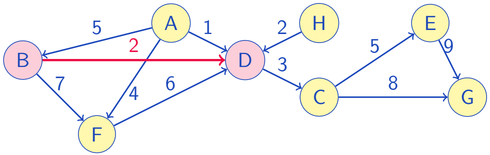


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	4	–	–	3	–	–	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	–	–	–	–	–	–	–
Value	1	2	3	4	–	–	–	–	–	–	–
Next	–	–	–	1	–	–	–	–	–	–	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

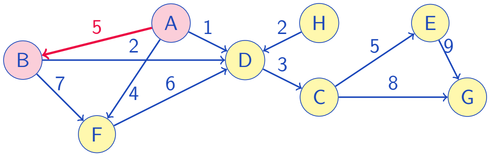


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	4	5	–	3	–	–	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	–	–	–	–	–	–
Value	1	2	3	4	2	–	–	–	–	–	–
Next	–	–	–	1	–	–	–	–	–	–	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant’s way: $O(1)$ dynamic data structures (outgoing only edges shown)

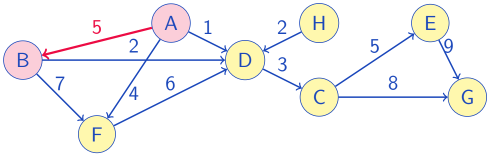


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	4	5	–	3	–	–	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	–	–	–	–	–	–
Value	1	2	3	4	2	–	–	–	–	–	–
Next	–	–	–	1	–	–	–	–	–	–	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

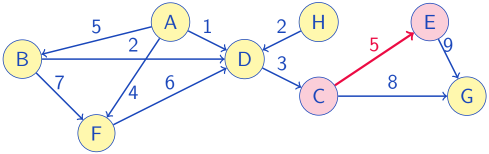


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	5	–	3	–	–	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	–	–	–	–	–
Value	1	2	3	4	2	5	–	–	–	–	–
Next	–	–	–	1	–	4	–	–	–	–	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

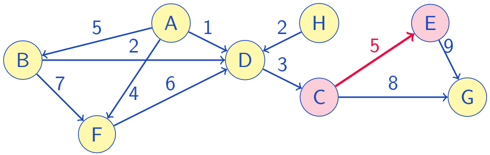


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	5	–	3	–	–	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	–	–	–	–	–
Value	1	2	3	4	2	5	–	–	–	–	–
Next	–	–	–	1	–	4	–	–	–	–	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant’s way: $O(1)$ dynamic data structures (outgoing only edges shown)

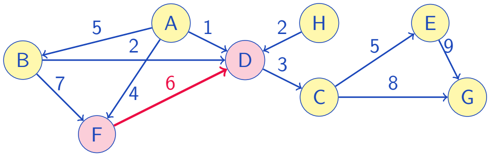


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	5	7	3	–	–	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	–	–	–	–
Value	1	2	3	4	2	5	5	–	–	–	–
Next	–	–	–	1	–	4	–	–	–	–	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

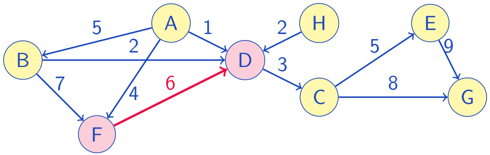


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	5	7	3	–	–	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	–	–	–	–
Value	1	2	3	4	2	5	5	–	–	–	–
Next	–	–	–	1	–	4	–	–	–	–	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant’s way: $O(1)$ dynamic data structures (outgoing only edges shown)

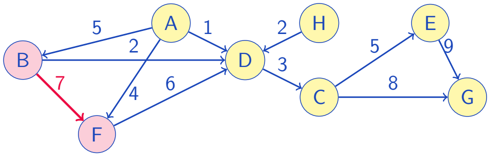


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	5	7	3	–	8	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	D	–	–	–
Value	1	2	3	4	2	5	5	6	–	–	–
Next	–	–	–	1	–	4	–	–	–	–	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

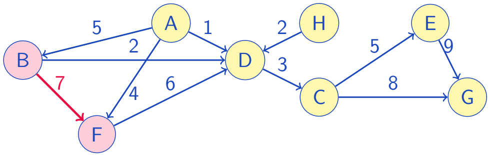


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	5	7	3	–	8	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	D	–	–	–
Value	1	2	3	4	2	5	5	6	–	–	–
Next	–	–	–	1	–	4	–	–	–	–	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)



- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

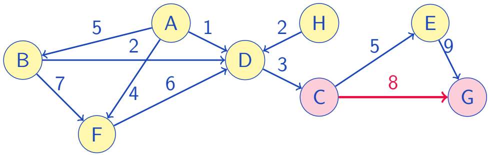
Vertex	A	B	C	D	E	F	G	H
Next	6	9	7	3	–	8	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	D	F	–	–
Value	1	2	3	4	2	5	5	6	7	–	–
Next	–	–	–	1	–	4	–	–	5	–	–

A compact storage for sparse graphs.

For every vertex, store incoming and outgoing edges.

The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

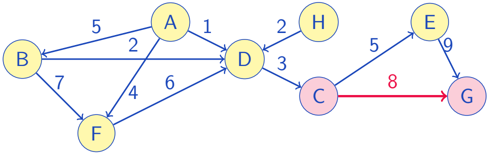


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	9	7	3	–	8	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	D	F	–	–
Value	1	2	3	4	2	5	5	6	7	–	–
Next	–	–	–	1	–	4	–	–	5	–	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

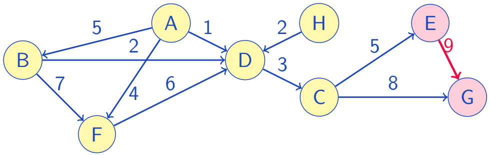


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	9	10	3	–	8	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	D	F	G	–
Value	1	2	3	4	2	5	5	6	7	8	–
Next	–	–	–	1	–	4	–	–	5	7	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant’s way: $O(1)$ dynamic data structures (outgoing only edges shown)

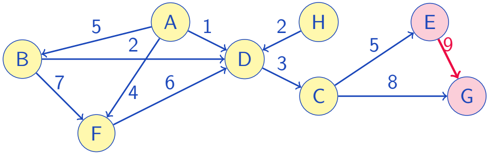


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	9	10	3	–	8	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	D	F	G	–
Value	1	2	3	4	2	5	5	6	7	8	–
Next	–	–	–	1	–	4	–	–	5	7	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

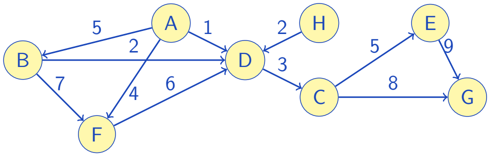


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	9	10	3	11	8	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	D	F	G	G
Value	1	2	3	4	2	5	5	6	7	8	9
Next	–	–	–	1	–	4	–	–	5	7	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant’s way: $O(1)$ dynamic data structures (outgoing only edges shown)

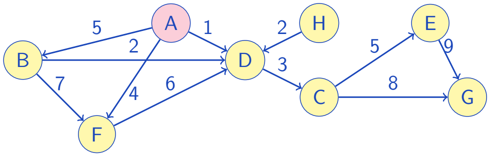


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	9	10	3	11	8	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	D	F	G	G
Value	1	2	3	4	2	5	5	6	7	8	9
Next	–	–	–	1	–	4	–	–	5	7	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

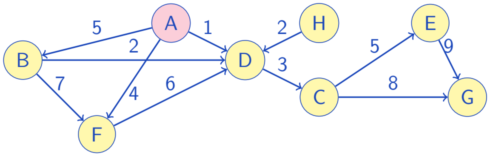


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	9	10	3	11	8	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	D	F	G	G
Value	1	2	3	4	2	5	5	6	7	8	9
Next	–	–	–	1	–	4	–	–	5	7	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

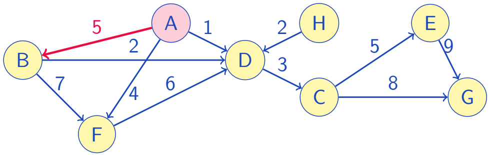


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	9	10	3	11	8	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	D	F	G	G
Value	1	2	3	4	2	5	5	6	7	8	9
Next	–	–	–	1	–	4	–	–	5	7	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant’s way: $O(1)$ dynamic data structures (outgoing only edges shown)

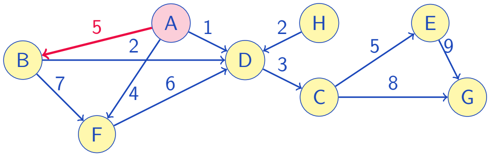


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	9	10	3	11	8	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	D	F	G	G
Value	1	2	3	4	2	5	5	6	7	8	9
Next	–	–	–	1	–	4	–	–	5	7	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

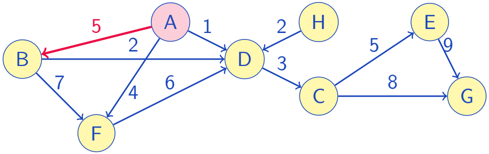


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	9	10	3	11	8	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	D	F	G	G
Value	1	2	3	4	2	5	5	6	7	8	9
Next	–	–	–	1	–	4	–	–	5	7	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant’s way: $O(1)$ dynamic data structures (outgoing only edges shown)

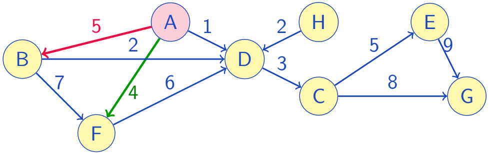


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	9	10	3	11	8	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	D	F	G	G
Value	1	2	3	4	2	5	5	6	7	8	9
Next	–	–	–	1	–	4	–	–	5	7	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant’s way: $O(1)$ dynamic data structures (outgoing only edges shown)

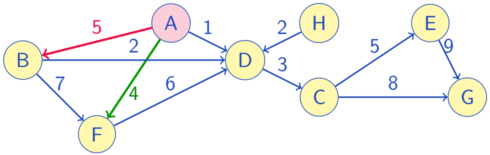


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	9	10	3	11	8	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	D	F	G	G
Value	1	2	3	4	2	5	5	6	7	8	9
Next	–	–	–	1	–	4	–	–	5	7	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

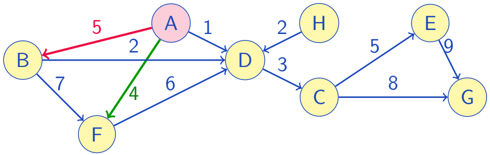


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	9	10	3	11	8	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	D	F	G	G
Value	1	2	3	4	2	5	5	6	7	8	9
Next	–	–	–	1	–	4	–	–	5	7	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)

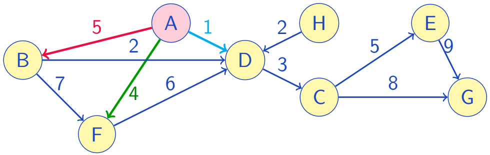


- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	9	10	3	11	8	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	D	F	G	G
Value	1	2	3	4	2	5	5	6	7	8	9
Next	–	–	–	1	–	4	–	–	5	7	–

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant's way: $O(1)$ dynamic data structures (outgoing only edges shown)



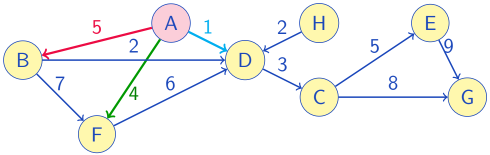
- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	9	10	3	11	8	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	D	F	G	G
Value	1	2	3	4	2	5	5	6	7	8	9
Next	–	–	–	1	–	4	–	–	5	7	–

Adjacency list: Alternative “old school” implementation

A compact storage for sparse graphs.
For every vertex, store incoming and outgoing edges.
The old contestant’s way: $O(1)$ dynamic data structures (outgoing only edges shown)



- Space requirements: $\Theta(|V| + |E|)$
- Edge addition: $\Theta(1)$ (amortized)
- Vertex addition: $\Theta(1)$ (amortized)
- Edge lookup/removal: $O(deg(v))$

Vertex	A	B	C	D	E	F	G	H
Next	6	9	10	3	11	8	–	2

Index	1	2	3	4	5	6	7	8	9	10	11
Vertex	D	D	C	F	D	B	E	D	F	G	G
Value	1	2	3	4	2	5	5	6	7	8	9
Next	–	–	–	1	–	4	–	–	5	7	–

- Adjacency matrix:
  - Space complexity: $\Theta(|V|^2)$
  - Perfect edge access and modification time: $\Theta(1)$
  - Good for storing dense graphs (say $|V| \approx 5000$, $|E| \approx 10\,000\,000$)
  - Good for working with transitive relations
  - Good for bitmask optimizations
  - Bad at iterating over vertex's adjacent edges: $\Theta(|V|)$

- Adjacency matrix:
  - Space complexity: $\Theta(|V|^2)$
  - Perfect edge access and modification time: $\Theta(1)$
  - Good for storing dense graphs (say $|V| \approx 5000$, $|E| \approx 10\,000\,000$)
  - Good for working with transitive relations
  - Good for bitmask optimizations
  - Bad at iterating over vertex's adjacent edges: $\Theta(|V|)$
- Adjacency list:
  - Space complexity: $\Theta(|V| + |E|)$
  - Edge access: $O(deg(v))$, or $O(\log(deg(v)))$ with binary trees
    - But trees require more memory (by a constant factor)!
  - Good for storing sparse graphs (say $|V| \approx 100\,000$, $|E| \approx 500\,000$)
  - Good at iterating over vertex's adjacent edges: $O(deg(v))$

- Adjacency matrix:
  - Space complexity: $\Theta(|V|^2)$
  - Perfect edge access and modification time: $\Theta(1)$
  - Good for storing dense graphs (say $|V| \approx 5000$, $|E| \approx 10\,000\,000$)
  - Good for working with transitive relations
  - Good for bitmask optimizations
  - Bad at iterating over vertex's adjacent edges: $\Theta(|V|)$
- Adjacency list:
  - Space complexity: $\Theta(|V| + |E|)$
  - Edge access: $O(deg(v))$, or $O(\log(deg(v)))$ with binary trees
    - But trees require more memory (by a constant factor)!
  - Good for storing sparse graphs (say $|V| \approx 100\,000$, $|E| \approx 500\,000$)
  - Good at iterating over vertex's adjacent edges: $O(deg(v))$
- Choose between them wisely!