Elementary Data Structures and Introduction to Graph Algorithms

Data Structures and Algorithms (094224)

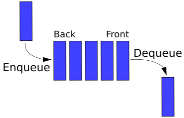
Tutorial 2

Winter 2022/23

Queue

Queue ⟨תור⟩

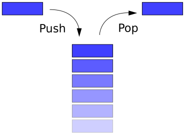
- First in first out (FIFO) policy
- Data structure's attributes:
  - $L =$ a (doubly) linked list, for implementation purposes only
  - $tail =$ pointer to last object of L, for implementation purposes only
- Can be implemented based on an array too
  - Bounded size
- Queue operations:
  - $\texttt{Enqueue}(Q, x)$ – Insert object x at the tail of queue Q
  - $\texttt{Dequeue}(Q)$ – Remove the object at the head of queue Q and return it



Stack

Stack ⟨מחסנית⟩

- Last in first out (LIFO) policy
- Data structure's attribute:
  - $L =$ a (doubly) linked list, for implementation purposes only
- Can be implemented based on an array too
  - Bounded size
- Stack Operations:
  - $\texttt{Push}(S, x)$ – Insert object x at the top of stack S
  - $\texttt{Pop}(S)$ – Remove the object at the top of stack S and return it



Question 1

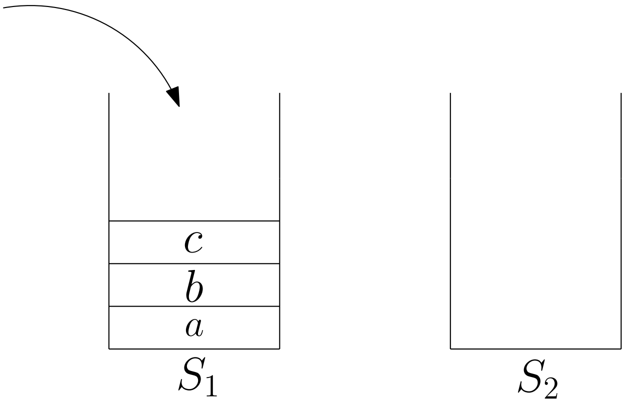
Show a way to implement a queue using two stacks and analyze the run time of the basic queue operations.

Solution:

- Data structure's attributes:
  - $S_1 =$ stack
  - $S_2 =$ stack
- Stack implementation assumptions:
  - Implementation allows finite but unbounded amount of elements
  - Stack operations run time is $O(1)$
- In order to implement a queue we need to implement:
  - $\texttt{Enqueue}(Q, x)$
  - $\texttt{Dequeue}(Q)$

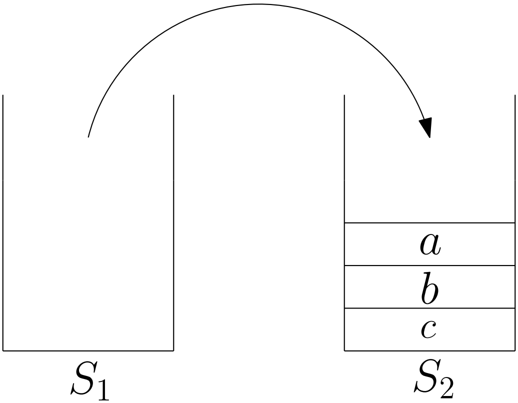
Solution — Example

`Enqueue`(Q, x):



Solution — Example

`Dequeue`(Q):



Solution

Enqueue(Q, x)

Push($Q.S_1, x$)

Dequeue(Q)

$y =$ Pop($Q.S_2$)
if $y \neq$ ERROR **then**
    return y
$y =$ Pop($Q.S_1$)
if $y ==$ ERROR **then**
    return ERROR
while $y \neq$ ERROR **do**
    Push($Q.S_2, y$)
    $y =$ Pop($Q.S_1$)
return Pop($Q.S_2$)

Solution — cont.

What is the running time?

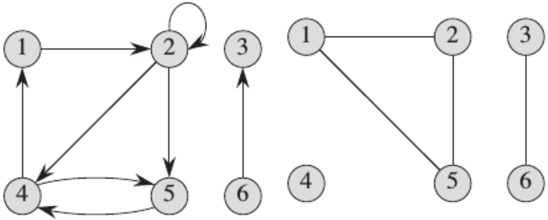
- $T_{\texttt{Enqueue}}(n) = O(1)$
- $T_{\texttt{Dequeue}}(n) = O(n)$, where n represents the number of elements in the queue
    - At most $O(n)$ calls to stack push and pop

1 Elementary data structures

2 Graph theory — basic definitions

3 Graph representation

- *Graph* ⟨גרף⟩ $G = (V, E)$
  - V = a set of *vertices* ⟨צמתים⟩ (or *nodes* ⟨קודקודים⟩)
  - E = a set of vertex pairs referred to as *edges* ⟨קשתות⟩ (or *arcs* ⟨צלעות⟩)
  - Unless stated otherwise, our graphs are *finite* ⟨סופיים⟩: V, E finite sets



Graph theory — basic definitions

- Directed vs. Undirected
- Incidence, adjacency, and degrees
- Path,cycle
- Connectivity - connected and strongly connected components
- Subgraph
- Important classes of undirected graphs: complete, bipartite, forest, tree
- Important classes of directed graphs: DAG

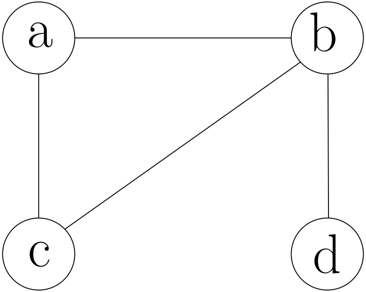
Question 2 — Handshake Lemma

The participants of a party shake hands as a form of greeting. During the party, each participant counts the number of handshakes he/she has given. After the party we sum all the individual handshake counts of the participants.

Show that the result is even.

Solution

- The setting described can be modeled as an undirected graph
  - Every participant is modeled as a vertex
  - Every handshake between two participants modeled as an edge between the corresponding vertices



Solution

- The individual handshake count of a participant is equivalent to the degree of the vertex representing the participant
- We therefore need to show that the sum of degrees is even
- In the lecture we have seen the degree sum formula

$$\sum_{v \in V} \deg(v) = 2|E|$$

- Since the result of multiplying any integer by 2 is an even number we are done

Question 3

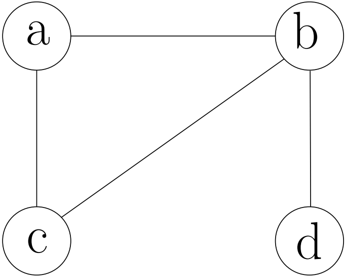
Given an undirected graph $G = (V, E)$ define the following:

- $U \subseteq V$ is a vertex cover of V if for every $(u, v) \in E$ it holds $u \in U \vee v \in U$
  - I.e., every edge of G has at least one endpoint in U
- $VC(G) =$ the size of a smallest vertex cover in G
- $U \subseteq V$ is an independent set if no two nodes in U are adjacent
  - I.e., for every $u, v \in U$ it holds $(u, v) \notin E$
- $IS(G) =$ the size of a largest independent set in G

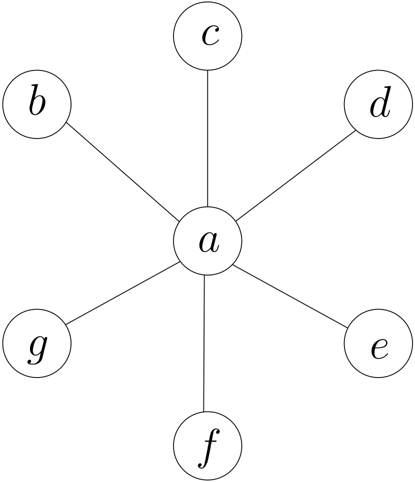
Prove: For every undirected graph $G = (V, E)$, it holds that $VC(G) = |V| - IS(G)$

Examples

G_1:



G_2:



Solution

Lemma

Let $G = (V, E)$ be an undirected graph. The set $U \subseteq V$ is a vertex cover iff $W = V - U$ is an independent set

Proof.

- Direction $\Longrightarrow$
- Assume by contradiction that W is not an independent set, i.e., there exist $u, v \in W$ such that $(u, v) \in E$
- By the definition of W it holds $u, v \notin U$
- The edge (u, v) is not covered $\Longrightarrow$ U is not a vertex cover $_{(\rightarrow\leftarrow)}$
- Direction $\Longleftarrow$
- Assume by contradiction $U = V - W$ is not a vertex cover i.e. there exists $e = (v, u) \in E$ such that $u, v \notin U$
- By the definition of U it holds $u, v \in W$
- Since $(u, v) \in E \Longrightarrow W$ is not an independent set $_{(\rightarrow\leftarrow)}$

Solution — cont.

Proof.

- Assume by contradiction that there exists an undirected graph $G = (V, E)$ such that $VC(G) \neq |V| - IS(G)$
- Without loss of generality
  Assume w.l.o.g. that $VC(G) > |V| - IS(G)$
- Let $W \subseteq V$ be an independent set such that $|W| = IS(G)$
- Let $U = V - W$ and notice that $|U| = |V - W| = |V| - |W| = |V| - IS(G) < VC(G)$
- By the Lemma, the set U is a vertex cover $_{(\rightarrow\leftarrow)}$

Representing graph in a computer

- How do we represent (un)directed graph $G = (V, E)$ in a computer?
  - Prerequisite for graph algorithms
- Conventions:
  - $n = |V|$
    - inside asymptotic notation, sometimes V
  - $m = |E|$
    - inside asymptotic notation, sometimes E
  - In pseudocode, we write $G.V$ and $G.E$ (attributes of G)
- Convenient to assume that the vertices are numbered $1, \ldots, n$
  - Arbitrary order
- Sometimes, vertices/edges have additional attributes (should be stored as well)
  - Common: edge *weights* ⟨משקלים⟩ $w : E \to \mathbb{R}$
    - *Weighted graph* ⟨גרף ממושקל⟩

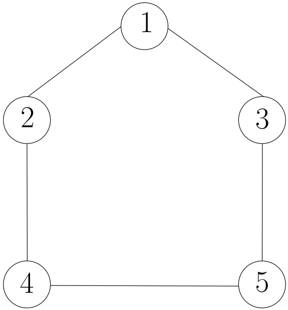
Adjacency matrix

- Matrix $A \in \{0,1\}^{n\times n}$
  - $A(i,j) = \begin{cases} 1, & (i,j) \in E \\ 0, & \text{otherwise} \end{cases}$
  - Called *adjacency matrix* ⟨מטריצת שכנויות⟩
- Advantage: can answer the query *"does* $(i,j) \in E$*?"* in $\Theta(1)$ time
- If the graph is weighted, then $A \in \mathbb{R}^{n\times n}$
  - Depending on the context, $(i,j) \notin E$ represented by $A(i,j) = 0$, $A(i,j) = +\infty$, or $A(i,j) = -\infty$
- Entry $A(i,j)$ can include a pointer to an object with additional attributes of edge (i,j)
- How much memory does the adjacency matrix representation require?

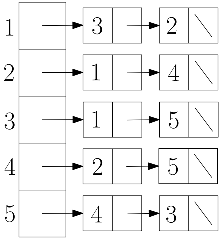
Adjacency lists

- Array *Adj* of n lists
- For each $u \in V$, $Adj[u]$ is an *adjacency list* ⟨רשימת שכנויות⟩ that contains an entry for each vertex $v \in V$ such that $(u, v) \in E$
  - The entry of v includes all attributes of edge (u, v)
  - Typically implemented with a (doubly) linked list
- Directed graph: edge (u, v) represented only in $Adj[u]$
  Undirected graph: edge (u, v) represented in $Adj[u]$ and $Adj[v]$
  - In the directed case, can include a list for the entering edges too if necessary
- Advantage: can enumerate the edges incident on (resp., from) vertex v in $\Theta(\deg(v))$ (resp., $\Theta(\deg_{\text{out}}(v))$ time
- How much memory does the adjacency list representation require?

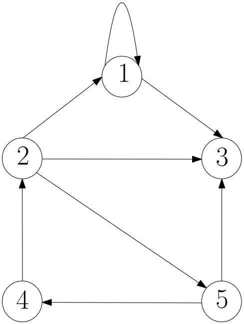
Example – undirected graph

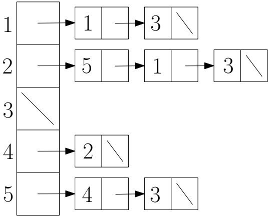


	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	0
3	1	0	0	0	1
4	0	1	0	0	1
5	0	0	1	1	0



Example – directed graph

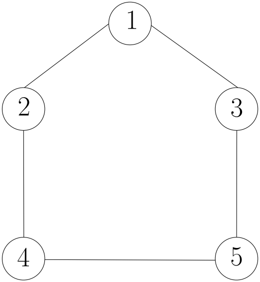




	1	2	3	4	5
1	1	0	1	0	0
2	1	0	1	0	1
3	0	0	0	0	0
4	0	1	0	0	0
5	0	0	1	1	0

Question 4

Given the following undirected graph $G = (V, E)$ and the adjacency matrix representation of G



	1	2	3	4	5
1	0	1	1	0	0
2	1	0	0	1	0
3	1	0	0	0	1
4	0	1	0	0	1
5	0	0	1	1	0

Calculate A^2 and describe what each of its elements represents.

Solution

$$A^2 = \begin{array}{c|c|c|c|c|c|} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 2 & 0 & 0 & 1 & 1 \\ \hline 2 & 0 & 2 & 1 & 0 & 1 \\ \hline 3 & 0 & 1 & 2 & 1 & 0 \\ \hline 4 & 1 & 0 & 1 & 2 & 0 \\ \hline 5 & 1 & 1 & 0 & 0 & 2 \\ \hline \end{array}$$

- $A^2(i,j) = A(i,:) \cdot A(:,j)$
  - Row i multiplied by column j

$$A^2(1,4) = \begin{array}{cc} 1 & \begin{pmatrix} 0 & 1 & 1 & 0 & 0 \end{pmatrix} \end{array} \begin{array}{c} 4 \\ \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 1 \end{pmatrix} \end{array} = 1$$

- $A^2(i,j) = \#$ length 2 paths starting at vertex i and ending at vertex j
- $A^2(i,i) = deg(i)$

Question 5

Given an adjacency lists representation of a **directed** graph, design the most efficient algorithm that you can to compute:

1. The sum of in-degrees
2. The out-degree of every vertex
3. The in-degree of every vertex

Solution – in-degrees

In order to compute the sum of in-degrees notice that for every directed graph $G = (V, E)$: $\sum_{v \in V} deg_{out}(v) = \sum_{v \in V} deg_{in}(v)$

In_Degrees(G)

```
count=0
for each v ∈ G.V do
    for each u ∈ G.Adj[v] do
        count += 1
return count
```

Solution – cont.

Complexity Analysis:

- For **every** graph G:
  $T_{\texttt{In\_Degrees}}(G) \leq \sum_{v \in V} [O(1) + O(1) deg_{out}(v)] = O(V + E)$
- Can we show $T_{\texttt{In\_Degrees}}(G) = \Omega(V + E)$?
  - Yes, not only there exists an infinite series of graphs for which
    $T_{\texttt{In\_Degrees}}(G) = \Omega(V + E)$, **for every** graph G
    $T_{\texttt{In\_Degrees}}(G) = \Omega(V + E)$

Adjacency Matrix:

- What will be the complexity if G is represented with an adjacency matrix?
  - Must explore each entry in adjacency matrix — $\Omega(V^2)$

Can we do better?

- Intuitively it seems hopeless (NOT a formal proof)

Solution — out-degree

- In order to compute each vertex out-degree we can follow each vertex adjacency list and count the length of the list

Out_Degree(G)

new array $OUT[1 \dots n]$
for all $v \in G.V$ **do**
    $OUT[v] = 0$
    for all $u \in G.Adj[v]$ **do**
        $OUT[v] += 1$
return OUT

Solution — cont.

Complexity Analysis:

- For **every** graph G:
  $T_{\texttt{Out\_Degree}}(G) \leq \sum_{v \in V}[O(1) + O(1)deg_{out}(v)] = O(V + E)$
- Can we show $T_{\texttt{Out\_Degree}}(G) = \Omega(V + E)$?
  - Yes, not only there exists an infinite series of graphs for which $T_{\texttt{Out\_Degree}}(G) = \Omega(V + E)$, **for every** graph G
    $T_{\texttt{Out\_Degree}}(G) = \Omega(V + E)$

Adjacency Matrix:

- What will be the complexity if G is represented with an adjacency matrix?
  - Must explore each entry in adjacency matrix — $\Omega(V^2)$

Can we do better?

- Intuitively it seems hopeless (NOT a formal proof)

Solution — in-degree

- In order to compute each vertex in-degree we can follow each vertex adjacency list and count the number of encounters in each vertex

```
In_Degree(G)
new array IN[1 ... n]
for all v ∈ G.V do
    IN[v] = 0
for all v ∈ G.V do
    for all u ∈ G.Adj[v] do
        IN[u] += 1
return IN
```

Solution — cont.

Complexity Analysis:

- For **every** graph G:
  $T_{\texttt{In\_Degree}}(G) \le \sum_{v\in V} O(1) + \sum_{v\in V}[O(1) + O(1)deg_{out}(v)] = O(V+E)$
- Can we show $T_{\texttt{In\_Degree}}(G) = \Omega(V+E)$?
  - Yes, not only there exists an infinite series of graphs for which $T_{\texttt{In\_Degree}}(G) = \Omega(V+E)$, **for every** graph G, $T_{\texttt{In\_Degree}}(G) = \Omega(V+E)$

Adjacency Matrix:

- What will be the complexity if G is represented with an adjacency matrix?
  - Must explore each entry in adjacency matrix — $\Omega(V^2)$

Can we do better?

- Intuitively it seems hopeless (NOT a formal proof)

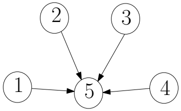
Question 6

Most graph algorithms that take an adjacency-matrix representation as input require time $\Omega(V^2)$, but there are some exceptions.

Show how to determine whether a **simple** directed graph G contains a **universal sink** - a vertex with in-degree $|V| - 1$ and out-degree 0 in time $O(V)$ given an adjacency matrix for G.

Solution — first attempt

- Keep a list L of all vertices with out-degree 0
  - A vertex is added to the list if all entires in row are equal to 0
- For every vertex in L check if all entries in the corresponding column are 1 (except diagonal element)
- What is the running time of the proposed algorithm ?
  - The algorithm is $O(V^2)$ since the algorithm may need to explore each cell in the matrix at most twice
  - Can we show that it is $\Omega(V^2)$?

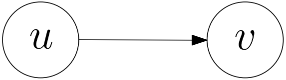


	1	2	3	4	5
1	0	0	0	0	1
2	0	0	0	0	1
3	0	0	0	0	1
4	0	0	0	0	1
5	0	0	0	0	0

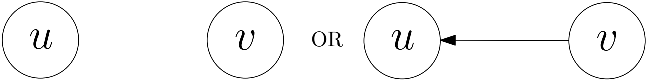
- Is there a more efficient way? What are the characteristics of the problem that can be exploited?

Solution

- For every pair of vertices $u, v \in V$
  - If $(u, v) \in E$, then u is not a universal sink



    - $\deg_{out}(u) \geq 1$
  - Otherwise, v is not a universal sink



    - $\deg_{in}(v) < n - 1$
- Only 0 or 1 sinks are possible

Solution

Input: simple directed graph G represented as adjacency matrix
Output: "true" if there exists a universal sink; "false" otherwise

Universal_Sink(G)

```
i = 1
for j = 2 to n do
    if G.A[i][j] == 1 then          ▷ (i, j) ∈ E
        i = j
deg_in = 0, deg_out = 0
for j = 1 to n do
    deg_in += G.A[j][i]
    deg_out += G.A[i][j]
if deg_in == n − 1 AND deg_out == 0 then
    return true
return false
```

Solution — cont.

Correctness:(NOT formal)

- At the end of the for loop in lines 2–4, only node i can be a universal sink:
  - Node k, $1 \leq k < i$ cannot be universal sink
    - Either, there exists $1 \leq j < k$ such that $G.A[j][k] = 0$ thus, $\deg_{in} < n - 1$; or
    - There exist $k < j \leq |V|$ such that $G.A[k][j] = 1$ thus, $\deg_{out} \geq 1$
  - Node k, $i < k \leq |V|$, cannot be universal sink
    - Since $G.A[i][k] = 0$ thus, $\deg_{in}(k) < n - 1$

Complexity analysis:

- Universal sink candidate found by performing $n - 1$ checks. After which we compute *in* and *out* degree of the candidate by performing the for loop in lines 6–8 n times. Thus, for **every** graph G: $T_{\texttt{Universal\_Sink}}(G) \leq O(n) + O(n) = O(n)$
- Can we show $T_{\texttt{Universal\_Sink}}(G) = \Omega(V)$?
  - Yes, not only there exists an infinite series of graphs for which $T_{\texttt{Universal\_Sink}}(G) = \Omega(V)$, **for every** graph G, $T_{\texttt{Universal\_Sink}}(G) = \Omega(V)$