BFS

Data Structures and Algorithms (094224)

Tutorial 4

Winter 2022/23

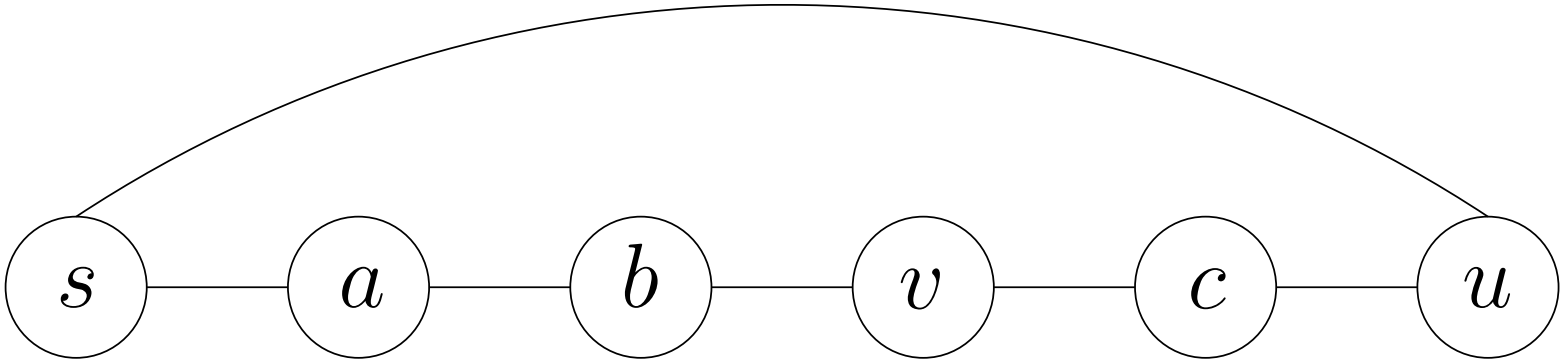
Warm up question — distance in a graph

Let $G = (V, E)$ be some directed or undirected graph.
Prove or disprove the following claims

1. Let $P = \langle s, a, b, v, c, u \rangle$ be some path in G. If $\delta(s, v) = 3$, then $\delta(s, u) = 5$.
2. If $P = \langle s, a, b, v, c, u \rangle$ is a shortest (s, u)-path in G, then $\delta(s, v) = 3$.
3. Let $P = \langle s, a, b, v, c, u \rangle$ be some path in G. If $\delta(s, u) = 5$, then $\delta(s, v) = 3$.
4. If $P = \langle s, a, b, v, c, u \rangle$ is a shortest (s, u)-path in G, then $\delta(s, u) = \delta(s, x) + \delta(x, u)$ for every $x \in \{s, a, b, v, c, u\}$.

Solution

1. False.



2. True.
   A subpath of a shortest path is a shortest path.
3. True.
   P is a shortest (s, u)-path
4. True.
   A subpath of a shortest path is a shortest path.

The breadth first search algorithm

breadth first search (BFS) ⟨סריקה לרוחב⟩

- One of the simplest graph algorithms
- Input:
  - A (di)graph $G = (V, E)$
  - A designated *source vertex* ⟨צומת מקור⟩ $s \in V$
- Explores G, discovering every vertex reachable from s
- Vertices discovered in order of their *distance* ⟨מרחק⟩ from s
  - Distance from $x \in V$ to $y \in V$ = length of a shortest (x, y)-path
    - Shortest path is not necessarily unique
- Constructs a *shortest path tree* ⟨עץ מסלולים קצרים ביותר⟩ rooted at s
  - Contains all vertices reachable from s
  - Unique simple (s, v)-path in the tree is a shortest (s, v)-path in G
  - a.k.a. *BFS tree*

Data structures

- $G = (V, E)$ represented using adjacency lists.
- Each vertex $v \in V$ maintains additional attributes:
  - $v.color$ = the discovery status of v
    - white = v is still undiscovered
    - gray = v has been discovered but it may still have undiscovered neighbors
    - black = v and all its neighbors have been discovered
  - $v.\pi$ = the parent of v in the BFS tree
  - $v.d$ = the distance from s to v in G
- Queue Q

Pseudocode

BFS(G, s)

1: *BFS_Initialization*(G, s, Q)
2: **while** $Q \neq \emptyset$ **do**
3: $u = \texttt{Dequeue}(\texttt{Q})$
4: **for all** $v \in G.Adj[u]$ **do**
5: **if** $v.color == white$ **then**
6: $v.color = gray$
7: $v.d = u.d + 1$
8: $v.\pi = u$
9: $\texttt{Enqueue}(Q, v)$
10: $u.color = black$

BFS_Initialization(G, s, Q)

1: **for all** $v \in G.V - \{s\}$ **do**
2: $v.color = white$
3: $v.d = \infty$
4: $v.\pi = NIL$
5: $s.color = gray$
6: $s.d = 0$
7: $s.\pi = NIL$
8: $Q = \emptyset$
9: $\texttt{Enqueue}(\texttt{Q}, \texttt{s})$

Question 1

The *diameter* ⟨קוטר⟩ of a graph $G = (V, E)$ denoted by $D(G)$ is defined as

$$D(G) = \max_{u,v \in V} \{\delta(u, v)\}.$$

Design as efficient as you can algorithm to compute the diameter of a tree $T = (V, E)$. Analyze its time complexity.

Inefficient Solution:

- For every $v \in V$ run $\texttt{BFS}(T, v)$
- After every run of $\texttt{BFS}(T, v)$ find $\max_{u \in V}\{u.d\}$
- Return the maximum value found
- Correctness: follows directly from correctness of BFS
  - Correct algorithm for any graph and not only trees
- Time complexity: $\Omega(V^2 + VE)$

Solution

Algorithm (Not a proper pseudocode):

- Input: A tree $T = (V, E)$
- Output: The diameter of T
  1. Pick an arbitrary vertex $s \in V$
  2. Run $\texttt{BFS}(T, s)$
  3. Let $u = \arg\max_{w \in V}\{w.d\}$
     - Break ties arbitrarily
  4. Run $\texttt{BFS}(T, u)$
  5. Return $\max_{w \in V}\{w.d\}$
     - $w.d$ refers to the value computed in $\texttt{BFS}(T, u)$

Solution — cont.

Time complexity:

- Pick an arbitrary vertex — $O(1)$
- $\texttt{BFS}(T, s)$ — $O(V + E)$
- Find u — $O(V)$
- $\texttt{BFS}(T, u)$ — $O(V + E)$
- Compute $\max_{w \in V}\{w.d\}$ — $O(V)$
- In total, the run time is $O(V + E)$
- Since the input is restricted to trees and for a tree $T = (V, E)$ it holds that $|E| = \Theta(V)$ the time complexity can be written as $O(V)$
- Notice: for algorithms with a general graph as input the complexity should depend on E and V

Solution — cont.

Correctness:

- Let T_s be the tree T rooted at s
- For every $x, y \in V$, define the lowest common ancestor of x and y in T_s as the deepest node that has both x and y as descendants
  - Denote the lowest common ancestor of x and y by $\ell(x, y)$
  - Notice: $\ell(x, y)$ exists for every $x, y \in V$ and it is unique
  - $\ell(x, y) = x$ if x is an ancestor of y (and vice versa)
- Observation 1: $\delta(x, y) = \delta(x, \ell(x, y)) + \delta(\ell(x, y), y)$ for all $x, y \in V$
  - Unique simple (x, y)-path goes through $\ell(x, y)$
- Let $u = \arg\max_{w \in V}\{\delta(s, w)\}$
  - Notice: u is a leaf of maximum depth in T_s
- Observation 2: $\delta(u, \ell(u, v)) \geq \delta(v, \ell(u, v))$ for all $v \in V$
  - Notice that $\delta(s, u) = \delta(s, \ell(u, v)) + \delta(\ell(u, v), u)$ and $\delta(s, v) = \delta(s, \ell(u, v)) + \delta(\ell(u, v), v)$
  - If $\delta(u, \ell(u, v)) < \delta(v, \ell(u, v))$, then $\delta(s, u) < \delta(s, v)$ which contradicts u's definition

Solution — cont.

- Observation 1: $\delta(x, y) = \delta(x, \ell(x, y)) + \delta(\ell(x, y), y)$ for all $x, y \in V$
- Observation 2: $\delta(u, \ell(u, v)) \geq \delta(v, \ell(u, v))$ for all $v \in V$
- Recall: we want to prove $\delta(x, y) \leq \max_{w \in V}\{\delta(u, w)\}$ for all $x, y \in V$
- Let $x, y \in V$ and assume w.l.o.g. that $\delta(u, \ell(x, u)) \leq \delta(u, \ell(y, u))$
- $\delta(u, y) = \delta(u, \ell(y, u)) + \delta(\ell(y, u), y)$ (Observation 1)
- $\delta(u, \ell(y, u)) = \delta(u, \ell(x, u)) + \delta(\ell(x, u), \ell(y, u))$
  - The simple $(u, \ell(y, u))$-path goes through all of u's ancestors between u and $\ell(y, u)$ including $\ell(x, u)$
- $\Rightarrow \delta(u, y) = \delta(u, \ell(x, u)) + \delta(\ell(x, u), \ell(y, u)) + \delta(\ell(y, u), y)$
- $\delta(x, y) \leq \delta(x, \ell(x, u)) + \delta(\ell(x, u), \ell(y, u)) + \delta(\ell(y, u), y)$
  - Length of any (x, y)-path bounds $\delta(x, y)$ from above
- $\delta(u, \ell(x, u)) \geq \delta(x, \ell(x, u))$ (Observation 2)
- $\Rightarrow \delta(x, y) \leq \delta(u, y) \leq \max_{w \in V}\{\delta(u, w)\}$ which concludes our proof

Question 2

Given an undirected connected graph $G = (V, E)$ design as efficient as you can algorithm to determine if G is a bipartite graph. Analyze its time complexity

Inefficient Solution:

- For every partition of G to V_1 and V_2 check if $E \subseteq V_1 \times V_2$
- #of partitions $= \Omega(2^n)$

Solution

Lemma A

Let $G = (V, E)$ be a bipartite graph with some partition V_1 and V_2 and let $P = \langle v, \cdots, u \rangle$ be a path. The length of P is even iff $(v, u \in V_1) \vee (v, u \in V_2)$

- G is bipartite, thus for every vertex $v \in V$ it holds: $(v \in V_1 \wedge v \notin V_2) \vee (v \in V_2 \wedge v \notin V_1)$
- An equivalent claim is: The length of P is odd iff $(v \in V_1 \wedge u \in V_2) \vee (v \in V_2 \wedge u \in V_1)$

Proof. By induction on the length of P

- Assume, w.l.o.g, $v \in V_1$
- Base: A path $P = \langle v, u \rangle$ of length 1
  - By definition of bipartite graph $u \in V_2$ since $(v, u) \in E \subseteq V_1 \times V_2$
  - For every $v \in V_1$ and $u \in V_2$, $P = \langle v, u \rangle$ is of length 1 (odd)

Solution — cont.

- Hypothesis: Let $P = \langle v, \cdots, u \rangle$ be a (v, u)-path of length $\ell - 1$
  - $\ell - 1$ is even iff $u \in V_1$
  - Equivalent to: $\ell - 1$ is odd iff $u \in V_2$
- Step:
  - Let $P = \langle v, \cdots, w, u \rangle$ be a (v, u)-path of length $\ell > 1$.
  - $(w, u) \in E$, thus $(w \in V_1 \wedge u \in V_2) \vee (w \in V_2 \wedge u \in V_1)$
    - By definition of bipartite graph $E \subseteq V_1 \times V_2$
  - The path $P' = \langle v, \cdots, w \rangle$ has length $\ell - 1$
  - $\Longrightarrow$ (by induction hypothesis) $w \in V_1$ iff $\ell - 1$ is even
  - $\Longrightarrow u \in V_2$ iff ℓ is odd

Solution — cont.

Lemma B

Let G be an undirected connected graph and let $s \in V$ be some arbitrary vertex. Graph G is bipartite iff for all $(u, v) \in E$, $(\delta(s, u)\ mod\ 2) \neq (\delta(s, v)\ mod\ 2)$

Solution:

- Direction $\Longrightarrow$
- Let V_1, V_2 be a partition of V (G is bipartite)
- Assume, w.l.o.g, $s \in V_1$
- According to Lemma A
  - For all $u \in V_1$ it holds $(\delta(s, u)\ mod\ 2) = 0$
  - For all $u \in V_2$ it holds $(\delta(s, u)\ mod\ 2) = 1$
- For all $(u, v) \in E$ it holds $(v \in V_1 \wedge u \in V_2) \vee (v \in V_2 \wedge u \in V_1)$
- If $(v \in V_1 \wedge u \in V_2)$, then $(\delta(s, v)\ mod\ 2) = 0 \neq (\delta(s, u)\ mod\ 2) = 1$
- Otherwise, $(\delta(s, v)\ mod\ 2) = 1 \neq (\delta(s, u)\ mod\ 2) = 0$

Solution — cont.

- Direction $\Longleftarrow$
- Let $V_1 = \{v \in V \mid (\delta(s, v) \, mod \, 2) = 0\}$
- Let $V_2 = \{v \in V \mid (\delta(s, v) \, mod \, 2) = 1\}$
- G is connected, thus $V_1 \cup V_2 = V$
- G is connected, thus for all $v \in V$, $\delta(s, v) < \infty$
- For all $v \in V$, $\delta(s, v)$ is unique thus $V_1 \cap V_2 = \emptyset$
- For all $(u, v) \in E$ it holds that $(\delta(s, v) \, mod \, 2) \neq (\delta(s, u) \, mod \, 2)$
- $\Longrightarrow (v \in V_1 \wedge u \in V_2) \vee (v \in V_2 \wedge u \in V_1)$
- $\Longrightarrow$ for all $(u, v) \in E$ it holds $(u, v) \in V_1 \times V_2$

Solution — cont.

Lemma B suggests the following algorithm for undirected connected graphs

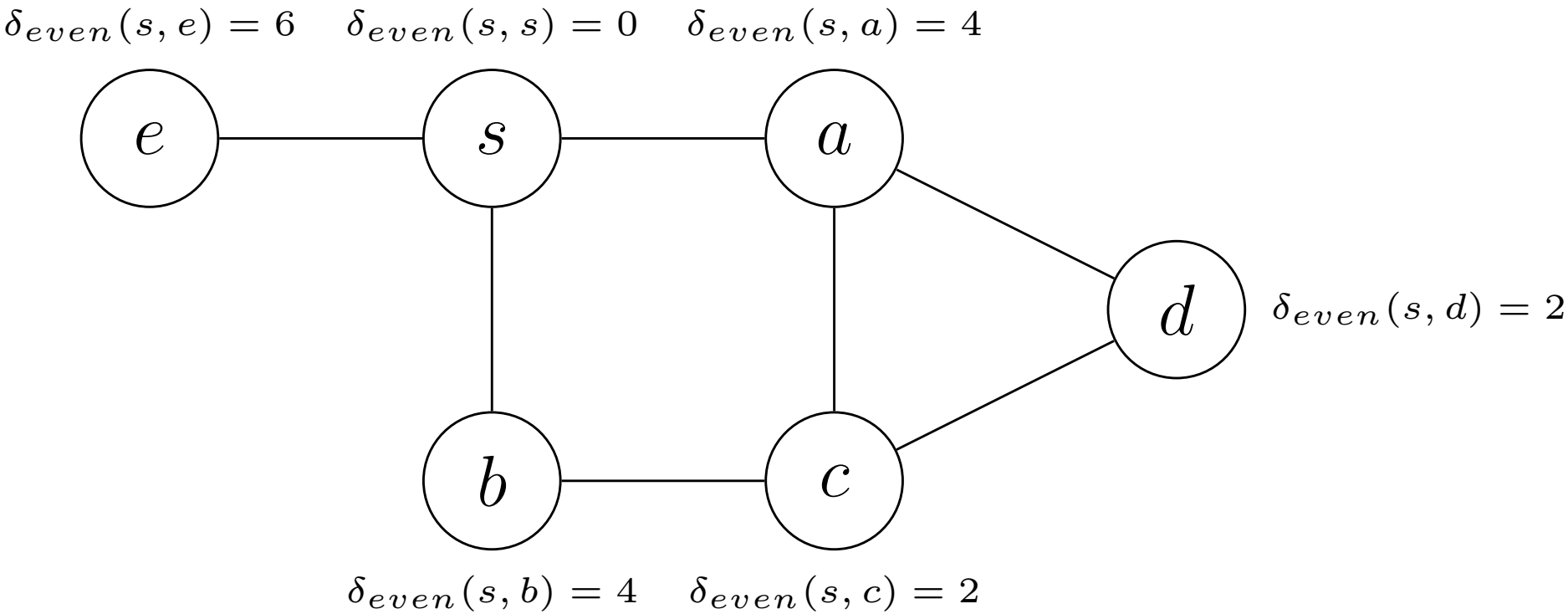
Algorithm (Not a proper pseudocode):

- Input: An undirected connected graph $G = (V, E)$
- Output: G is bipartite *true*/*false*
  1. Pick an arbitrary vertex $s \in V$
  2. Run $\texttt{BFS}(G, s)$
  3. If for all $(u, v) \in E$ it holds $(\delta(s, v)\ mod\ 2) \neq (\delta(s, u)\ mod\ 2)$ return *true* otherwise return *false*
- Correctness: Follows directly from Lemma B
- Time complexity:
  - Pick an arbitrary vertex — $O(1)$
  - $\texttt{BFS}(G, s)$ — $O(V + E)$
  - Scan all edges (adjacency list representation) — $O(V + E)$
  - In total, the run time is $O(V + E)$

Question 3

Given an undirected graph $G = (V, E)$ and a vertex $s \in V$, propose an algorithm that finds for every vertex $v \in V$ the length of the shortest path between s and v whose length is even (not necessarily a simple path) or ∞ if such a path does not exists. Prove correctness of the proposed algorithm and analyze its time complexity.

Question 3 — Example



Solution — 1

Variation of BFS

- The main idea of the algorithm that will be presented is to use a variation of BFS

Data Structures:

- $G = (V, E)$ represented using adjacency list
- Each vertex $v \in V$ maintain additional attributes
  - $v.d\_even$
  - $v.d\_odd$
  - $v.even\_color$
  - $v.odd\_color$
- Two queues Q_even and Q_odd

Solution — 1 — cont.

`Even_Paths`(G, s)

```
1:  even_queue = true
2:  for all u ∈ G.V − {s} do
3:      u.d_even = ∞
4:      u.d_odd = ∞
5:      u.even_color = white
6:      u.odd_color = white
7:  s.d_even = 0
8:  s.d_odd = ∞
9:  s.even_color = grey
10: s.odd_color = white
11: Q_even = ∅
12: Q_odd = ∅
13: Enqueue(Q_even, s)
14: while Q_even ≠ ∅ OR Q_odd ≠ ∅ do
15:     if even_queue == true then
16:         while Q_even ≠ ∅ do
17:             u = Dequeue(Q_even)
```

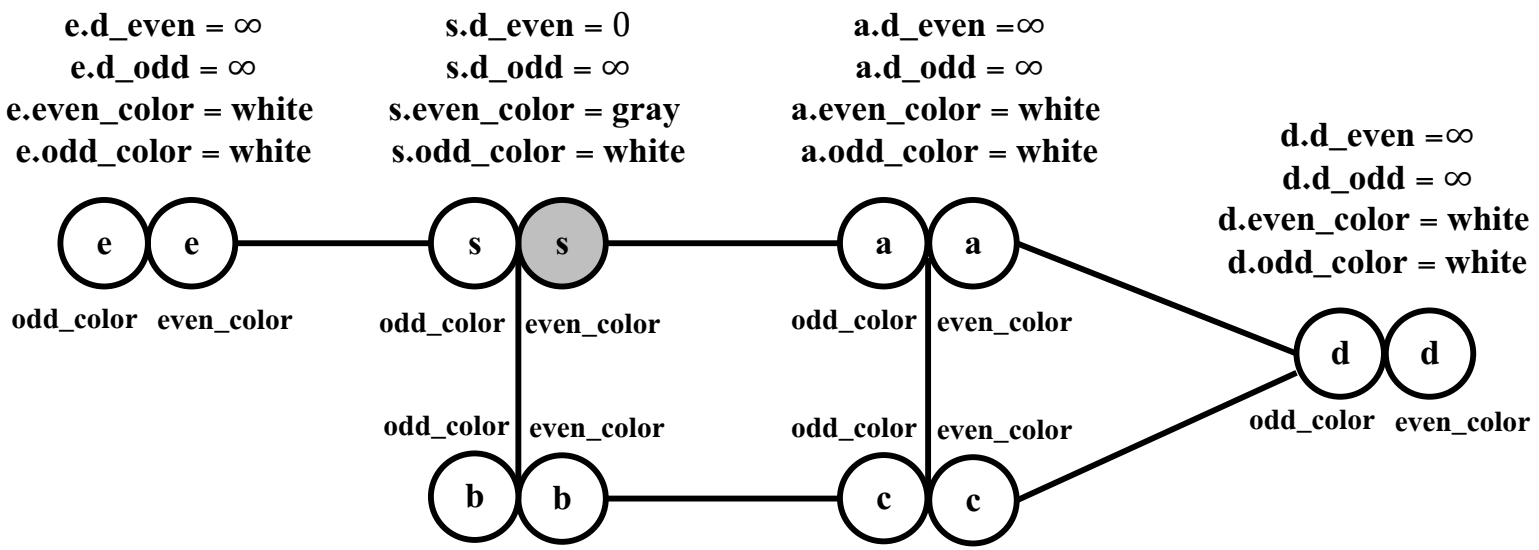
Solution — 1 — cont.

```
18:             for all v ∈ G.Adj[u] do
19:                 if v.odd_color == white then
20:                     v.odd_color = gray
21:                     v.d_odd = u.d_even + 1
22:                     Enqueue(Q_odd, v)
23:             u.even_color = black
24:         even_queue = false
25:     else
26:         while Q_odd ≠ ∅ do
27:             u = Dequeue(Q_odd)
28:             for all v ∈ G.Adj[u] do
29:                 if v.even_color == white then
30:                     v.even_color = gray
31:                     v.d_even = u.d_odd + 1
32:                     Enqueue(Q_even, v)
33:             u.odd_color = black
34:         even_queue = true
```

Solution — 1 — cont.

- We maintain two queues
- First we enqueue s in Q_even
- For every dequeued vertex v from Q_even we scan its adjacency list for undiscovered vertices
- For every undiscovered vertex $u \in adj[v]$ we update its odd path to s, color its odd_color attribute gray and enqueue to Q_odd
- After entire v's adjacency list is scanned v's $color\_even$ attribute is set to black
- Once Q_even gets empty we start dequeue from Q_odd
- For every dequeue vertex v from Q_odd we scan its adjacency list for undiscovered vertices
- For every undiscovered vertex $u \in adj[v]$ we update its even path to s, color its $even\_color$ attribute gray and enqueue to Q_even
- After entire v's adjacency list is scanned v's $color\_odd$ attribute is set to black
- This process continues until both queues are empty

Solution — 1 — cont.



Q_even (True) : s

Q_odd : $\emptyset$

Solution – 1 – Run-time analysis

- Initialization takes $O(n)$ time
- After initialization, *Even_Paths*(G, s) never colors a vertex white (odd color and even color)
- If vertex $v \in V$ passes the test in line 19 or 29, then it is colored gray in line 20 or 30 respectively
- $\Longrightarrow$ every vertex v is enqueued at most twice
- $\Longrightarrow$ every vertex v is dequeued at most twice
  - $O(n)$ time for the queue operations
- $\Longrightarrow$ Every adjacency list is traversed at most twice
- Total size of all adjacency lists is $O(n + m)$
- Each entry of each adjacency list accounts for $O(1)$ time
- $\Longrightarrow$ In total, the run-time of `Even_Paths`(G, s) is $O(n + m)$

Solution – 1 – Correctness – The Goal

- $\delta_{even}(x, y)$ = length of a shortest (x, y)-path with even number of edges
  - ∞ if such a path does not exists
- $\delta_{odd}(x, y)$ = length of a shortest (x, y)-path with odd number of edges
  - ∞ if such a path does not exists
- Goal: show that when `Even_Paths`(G, s) terminates:
  - $v.d\_even = \delta_{even}(s, v)$ for every vertex $v \in V$

Solution – 1 – Correctness — cont.

Lemma 1

If $(u, v) \in E$, then

- $\delta_{even}(s, v) \leq \delta_{odd}(s, u) + 1$
- $\delta_{odd}(s, v) \leq \delta_{even}(s, u) + 1$

Proof.

- Similar to lecture

Lemma 2

Upon termination of `Even_Paths`(G, s), $v.d\_even \geq \delta_{even}(s, v)$ and $v.d\_odd \geq \delta_{odd}(s, v)$ for all $v \in V$.

Proof.

- We show that the assertion holds immediately after each enqueue operation (Lines 22 and 32) by induction on #enqueue operations
- Base: Similar to lecture
- Step: Show for enqueue Q_odd and for enqueue Q_even

Solution — 1 — Correctness — cont.

Lemma 3

If during the execution $Q_{even} = \langle v_1, \ldots, v_r \rangle$, where v_1 is the head and v_r is the tail, and $Q_{odd} = \langle v_{r+1}, \ldots, v_k \rangle$, where v_{r+1} is the head and v_k is the tail then

- If $even\_queue = true$ then
  $v_1.d\_even \leq v_2.d\_even \leq \cdots \leq v_r.d\_even \leq v_1.d\_odd \leq \cdots \leq v_k.d\_odd \leq v_1.d\_even + 1$
- Else
  $v_{r+1}.d\_odd \leq v_{r+2}.d\_odd \leq \cdots \leq v_k.d\_odd \leq v_1.d\_even \leq \cdots \leq v_r.d\_even \leq v_{r+1}.d\_odd + 1$

Proof.

- We show that the assertion holds immediately after each queue operation (Lines 17, 22, 27 and 32) by induction on #queue operations
  - Similar to lecture, show for queue operation enqueue and dequeue based on the value of *even_queue*

Solution — 1 — Correctness — cont.

Corollary 1

1. If u was enqueued before v to Q_odd then at all times, $u.d\_odd \leq v.d\_odd$
2. If u was enqueued before v to Q_even then at all times, $u.d\_even \leq v.d\_even$
3. If u was enqueued to Q_even before v to Q_odd then at all times, $u.d\_even \leq v.d\_odd$
4. If u was enqueued to Q_odd before v to Q_even then at all times, $u.d\_odd \leq v.d\_even$

Proof.

- Follows directly from the last lemma and the fact that the d_even and d_odd attributes of a vertex are set once

Solution – 1 – Correctness — cont.

Theorem

Upon termination of $\texttt{Even\_Paths}(G, s)$, $v.d\_even = \delta_{even}(s, v)$ for every vertex $v \in V$

Proof.

- We are left to show $v.d\_even \leq \delta_{even}(s, v)$ for every vertex $v \in V$
- Assume by contradiction that upon termination of $\texttt{Even\_Paths}(G, s)$, $v.d\_even > \delta_{even}(s, v)$ for some vertex v with even length path from s
  - Clearly, $v \neq s$
- Take v to be such a vertex that minimizes $\delta_{even}(s, v)$
- Let z be the vertex preceding v along some shortest even length (s, v)-path in G
- Let u be the vertex preceding z along the same shortest even length (s, v)-path in G
  - Possibly $s = u$

Solution – 1 – Correctness — cont.

- $\delta_{even}(s, v) = \delta_{odd}(s, z) + 1$
- $\delta_{odd}(s, z) = \delta_{even}(s, u) + 1$
- By the choice of v, $u.d\_even = \delta_{even}(s, u)$ since $\delta_{even}(s, u) < \delta_{even}(s, v)$
- Use same arguments from lecture to prove $z.d\_odd \leq \delta_{even}(s, u) + 1 = \delta_{odd}(s, z)$
  - Consider time t when u was dequeued from Q_{even}
- Combine with Lemma 2 we get $z.d\_odd = \delta_{odd}(s, z)$
- Use same arguments from lecture and $z.d\_odd = \delta_{odd}(s, z)$ to prove $v.d\_even = \delta_{even}(s, v)$
  - Consider time t when z was dequeued from Q_{odd}

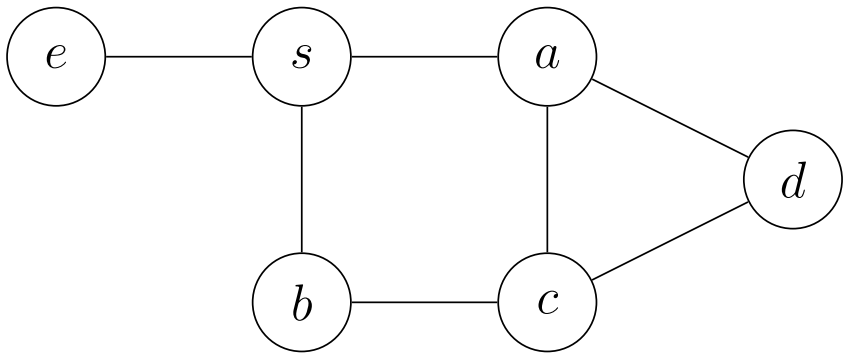
Solution — 2

Reduction

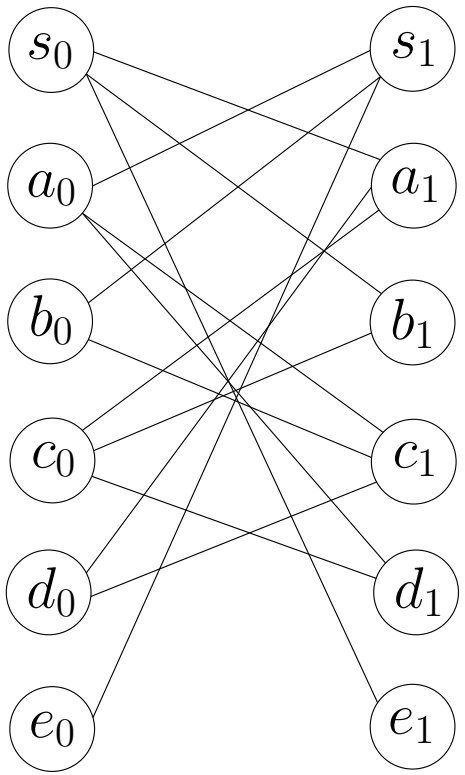
Given an undirected graph $G = (V, E)$ we construct a graph $G' = (V', E')$ in the following way:

- For every vertex $v \in V$ we create two vertices $v_0, v_1 \in V'$
- For every edge $(u, v) \in E$ we create two new edges $(u_0, v_1), (u_1, v_0) \in E'$

G :



G' :



Solution — 2 — cont.

- Notice G' is a bipartite graph by definition
- According to Lemma A: a path $P = \langle u, \ldots v \rangle$ between u and v in G' is of even length iff v and u are on the same side

Lemma 4

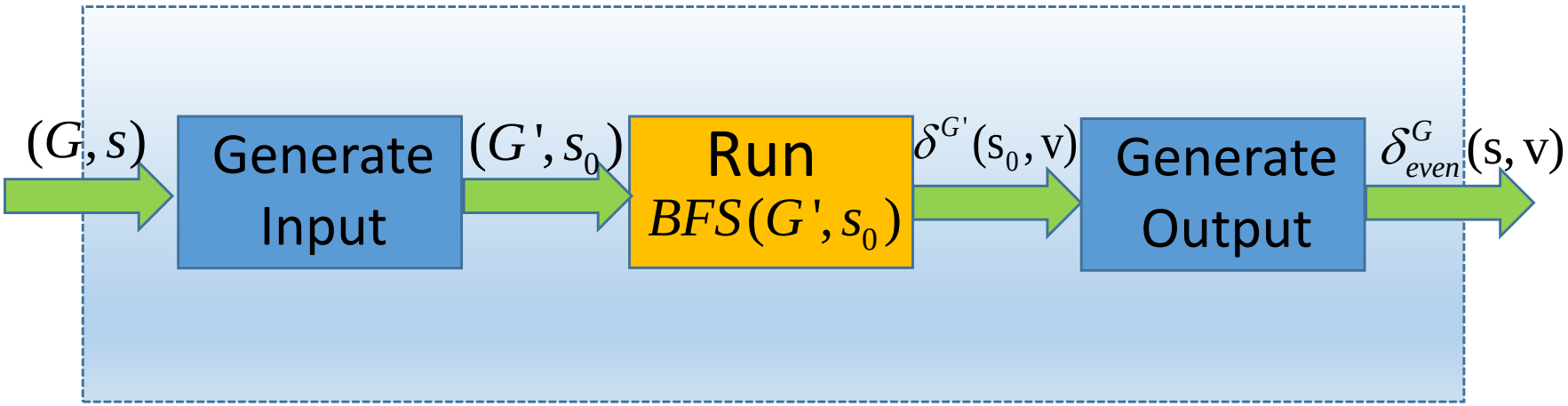
Let $G = (V, E)$ be an undirected graph and $s \in V$ then,
$\delta^G_{even}(s, v) = \delta^{G'}(s_0, v_0)$

Proof.

- There is a one to one mapping between even length paths from s_0 to v_0 in G' to even length paths from s to v in G for all $v \in V$

Solution — 2 — cont.

- We would like to use BFS algorithm "as is" (a black box) to solve our problem
- In order to solve question 3 with BFS as a black box we will change the input to BFS such that the output of BFS will be a solution to question 3
- This approach is called Reduction
  - By using a known algorithm, we don't have to prove correctness (only that our reduction is correct)
  - Be careful with complexity, the input to our problem is G while the input to BFS is G'



Solution — 2 — cont.

Algorithm (Not a proper pseudocode):

- Input: An undirected graph $G = (V, E)$ and a vertex $s \in V$
- Output: $\delta_{even}(s, v)$ for every $v \in V$
  1. Build $G' = (V', E')$ from $G = (V, E)$
  2. Run $\texttt{BFS}(G', s_0)$
  3. For every $v \in V$, $v.d = v_0.d (= \delta^{G'}(s_0, v_0) = \delta^{G}_{even}(s, v))$

Time Complexity:

- Constructing G' from G — $O(2V + 2E) = O(V + E)$
- Running BFS on G' — $O(2V + 2E) = O(V + E)$
- Updating the even distances — $O(V)$
- In Total, the run time is $O(V + E)$