Practice problems for the final

Claude Gravel

2024/04/15 — 12:05:13

Given a directed graph $G = (V, E)$ with $|V| = n > 0$, we want to determine is a vertex y is accessible from a vector x. This is usually known as a variant of Floyd algorithm, which is a dynamic programming algorithm. We number the vertices from 1 to n. Let $A(k) = \{1, 2, \ldots, k\}$ be the set of the first k vertices. By convention we let $A(0) = \emptyset$. Consider $c = (v_1, \ldots, v_q)$ a path of G. The **interior** $I(c)$ of the path c is the set of vertices of the sub-path $(v_2, \ldots, v_{q-1})$. When $q \leq 2$, then $I(c) = \emptyset$. Let $G_k = (S, E_k)$ be the graph defined by

$$(i, j) \in E_k \leftrightarrow \exists \text{ a path } c \text{ from } i \text{ to } j \text{ such that } I(c) \subset A(k).$$

Show (or, at least, convince yourself with examples) that $\forall i, j, k \in V$, we have that

$$(i, j) \in E_k \textbf{ if and only if } (i, j) \in E_{k-1} \text{ or } \big((i, k) \in E_k \text{ and } (k, j) \in E_k\big).$$

The previous fact about E_k suggest the following algorithm:

Algorithm 1 `Floyd_accessibility` (M (matrix))

Input: An adjacency matrix M of size n-by-n of a directed graph.
Output: The accessibility matrix D

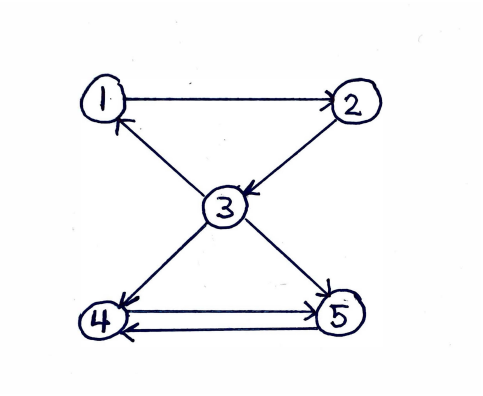
```
1:  D ← M                                          // Initialize boolean matrix D
2:  D[i, i] ← 1 for all i ∈ V                      // Any state is accessible to itself.
3:  for k from 1 to n do
4:      for i from 1 to n do
5:          for j from 1 to n do
6:              D[i, j] ← D[i, j] ∨ (D[i, k] ∧ D[k, j])
7:          end for
8:      end for
9:  end for
10: Return D
```

Question 1: Why Algorithm `Floyd_accessibility` fits in the dynamic programming approach? What is its worst-case time complexity?

Solution: Clearly it runs in $O(n^3)$. It is a dynamic programming method because the matrix D is a table that is filled through successive iteration (loop with k), and which stores all the possible ways to access a given vertex to another vertex whenever the former and the latter vertices are connected.

Question 2: Execute Algorithm `Floyd_accessibility` on the graph $G = (V, E)$ with $V = \{1, 2, 3, 4, 5\}$ and

$$E = \{(1,2), (2,3), (3,1), (3,4), (3,5), (4,5), (5,4)\}$$



Solution: By execution, we mean the representation of the dynamic programming table, that is D, at iteration $k = 1, \ldots, k = 5$. For simplicity, let's us denote by $D^{(k)}$ the table after iteration k is completed, and $D^{(0)} = D$ (the initial input). Then

$$D^{(1)} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

$$D^{(4)} = D^{(3)}$$

$$D^{(5)} = D^{(4)}$$

Let $G = (V, E)$ be a directed graph. We adapt Algorithm `Floyd_accessibility` to find the shortest path between two vertices with respect to the number of edges.

Let d_{ij} be the length of **a** shortest path between vertices i and j. Here d_{ij} defines an integer matrix. Consider a shortest path $c = (v_1, \ldots, v_k)$ with $v_1 = i$ and $v_k = j$. As in the previous exercise, let E_k be the set of vertices $\leq k$. Let $d_{ij}^{(k)}$ be the length of the shortest path between i and j with interior included in $E(k)$. Here $d_{ij}^{(k)}$ defines another integer matrix.

Question 3: What is $d_{ij}^{(0)}$?

Solution: We clearly have that $d_{ij}^{(0)} = 0$ if $i = j$ and $d_{ij}^{(0)} = \infty$ if $i \neq j$.

Question 4: What is the recurrence relation between $d_{ij}^{(k+1)}$ and $d_{ij}^{(k)}$. Then find a matrix (dynamic programming table) that contains all values d_{ij}.

Solution: We recall that the interior of a path $c = (v_1, s_2, v_{k-1}, v_k)$, denoted by $I(c)$, is the set of vertices of the sub-path $(v_2, \ldots, v_{k-1})$. If $k \leq 2$, then $I(c) = \emptyset$. Let c be the shortest path from i to j such that $I(c) \subset \{v_1, \ldots, v_k\}$. Let c' be a shortest path from i to v_{k+1} and c'' be a shortest path from v_{k+1} to j. We have $d_{i,k+1}^{(k)} = |c'|$ and $d_{k+1,j}^{(k)} = |c''|$. Therefore we have

$$d_{ij}^{(k+1)} = \min\{d_{ij}^{(k)}, d_{i,k+1}^{(k)} + d_{k+1,j}^{(k)}\}.$$

Question 5: Write an algorithm to find d_{ij}.

Solution: The major change with respect to Algorithm 1 is to replace line (6) with $d_{ij}^{(k+1)} \leftarrow \min\{d_{ij}^{(k)}, d_{i,k+1}^{(k)} + d_{k+1,j}^{(k)}\}$.

We study more variants of Floyd's algorithms. We are interested this time in *weighted connected directed* graph. Let $G = (V, E, \nu)$ where ν is a weight function. We find paths of minimal weight between any pair (i, j) of vertices in G. (*In this context*, we may simply talk about the shortest path problem.)

Definition: A circuit is *absorbing* if it is a circuit with a *strictly* negative weight. (When looking to maximal weight path, then a circuit is *absorbing* if it has a strictly positive weight.

Here is another algorithm *à la Floyd* to find all paths with minimal weights. Algorithm `Floyd_min_weight` requires *no* absorbing circuits to be correct.

Algorithm 2 `Floyd_min_weight` (M (matrix), ν (function))

Input: An adjacency matrix M of size n-by-n of a directed graph.
Input: A function $\nu : V \times V \to \mathbb{R}$. // This could be a table.
Output: A matrix D such that $(D)_{ij}$ is the weight of **a** minimal weight path between i and j.

1: $D \leftarrow \nu(i, j)$ if $(i, j) \in E$. // Initialize real matrix D
2: $D \leftarrow +\infty$ if $(i, j) \notin E$. // This can be done with an auxiliary matrix with flags.
3: **for** k **from** 1 **to** n **do**

```
    for i from 1 to n do
        for j from 1 to n do
            D[i, j] ← min{D[i, j] , D[i, k] + D[k, j]}
        end for
    end for
end for
Return D
```

Question 6: Execute Algorithm `Floyd_min_weight` on the graph with G with $V = \{1, 2, 3, 4, 5\}$,

$$E = \{(2,4), (3,2), (3,5), (4,1), (4,3), (4,5), (5,1)\},$$

and

$(i,j) \in E$	$\nu(i,j)$
$(2,4)$	3
$(3,2)$	-1
$(3,5)$	-7
$(4,1)$	1
$(4,3)$	8
$(4,5)$	1
$(5,1)$	2

Solution:

$$D^{(1)} = \begin{pmatrix} 0 & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & 3 & \infty \\ \infty & -1 & 0 & \infty & -7 \\ 1 & \infty & 8 & 0 & 1 \\ 2 & \infty & \infty & \infty & 0 \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & 3 & \infty \\ \infty & -1 & 0 & 2 & -7 \\ 1 & \infty & 8 & 0 & 1 \\ 2 & \infty & \infty & \infty & 0 \end{pmatrix}$$

$$D^{(3)} = \begin{pmatrix} 0 & \infty & \infty & \infty & \infty \\ \infty & 0 & \infty & 3 & \infty \\ \infty & -1 & 0 & 2 & -7 \\ 1 & 7 & 8 & 0 & 1 \\ 2 & \infty & \infty & \infty & 0 \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & \infty & \infty & \infty & \infty \\ 4 & 0 & 11 & 3 & 4 \\ 3 & -1 & 0 & 2 & -7 \\ 1 & 7 & 8 & 0 & 1 \\ 2 & \infty & \infty & \infty & 0 \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & \infty & \infty & \infty & \infty \\ 4 & 0 & 11 & 3 & 4 \\ -5 & -1 & 0 & 2 & -7 \\ 1 & 7 & 8 & 0 & 1 \\ 2 & \infty & \infty & \infty & 0 \end{pmatrix}$$

We recall:

1. Let $G = (V, E)$ be an undirected graph (resp. directed) such that $|V| = n$ and $|E| = m$, and let $U \subset V$.
2. The **boundary** of U, denoted by $B(U, G)$, is the set of vertices $V \setminus U$ (resp. successors) defined as follows

   $v \in B(U, G)$ if and only if $v \in V \setminus U$ and there is $u \in U$ such that $(u, v) \in E$.

3. A **generic path** starting at $v_{i_1} \in V$ is list $L = (v_{i_1}, v_{i_2}, \ldots, v_{i_n})$ representing a permutation of V *if and only if* for all $j \in \{2, \ldots, n\}$, the boundaries $B(L[1, j-1], G) \neq \emptyset \Longrightarrow v_{i_j} \in B(L[1, j-1], G)$ with $L[1, j-1]$ as a sub-list of the vertices $v_{i_1}, v_{i_2}, \ldots, v_{i_{j-1}}$.
4. A path L is a **breadth-first path** if for all $j \in \{2, \ldots, n\}$ such that $B(L[1, j-1], G) \neq \emptyset$, the vertex v_{i_j} is a neighbour (resp. a successor) of the *first* vertex from $L[1, j-1]$ with at least one adjacent vertex in $L[j, n]$.
5. A path L is a **depth-first path** if for all $j \in \{2, \ldots, n\}$ such that $B(L[1, j-1], G) \neq \emptyset$, the vertex v_{i_j} is a neighbour (resp. a successor) of the *last* vertex from $L[1, j-1]$ with at least one adjacent vertex in $l[j, n]$.

Let us denote by `ALGO_1` the following algorithm:

Algorithm 3 `ALGO_1`(L : list, G : graph)

Input: G: a graph (V, E)
Input: L: a list of size $|V| = n$ containing all the vertices of G
Output: Ψ: set of edges

```
1: i ← 1
2: j ← 2
3: Ψ ← ∅
4: while j ≤ n do
5:     if {v_i, v_j} ∈ E then
6:         Ψ ← Ψ ∪ {(v_i, v_j)}
7:         j ← j + 1
8:     else
9:         i ← i + 1
10:    end if
11: end while
12: Return Ψ
```

Question 7a: Let $G = (V, E)$ be the undirected simple connected graph with

$V = \{a, b, c, d, e, f, g\}$ and matrix representation given by

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

Row and columns indices follow the alphabetical order of the vertices. Let $L = \{a, g, c, b, f, d, e\}$ be a list. Explain why L represents a breadth-first path. First, the list L must contain all the vertices of G, and what else must L satisfy to represent a valid breadth-first path?

Solution: L is a valid breadth-first path because starting from a, the vertex v_i for $i \in \{2, 3, 4, 5, 6, 7\}$ is adjacent to the first visited vertex from $(v_1, \ldots, v_i)$.

Question 7b: What does `ALGO_1` compute on given input G and L from (**1a**)? (We have seen it in class.)

Solution: The breadth-first tree represented by L with respect to the graph G.

Question 7c: What does `ALGO_1` return on inputs G and L from (**1a**).

Solution: $\{(a, g), (a, c), (a, b), (g, f), (c, d), (b, e)\}$

An algorithm for boundaries

Question 8: Let $G = (V, E)$ be an undirected graph with $V = \{1, 2, \ldots, n\}$, and $|E| = m$. Give an algorithm, say `Boundary`, which takes a list L, a graph G and an index j and returning the boundary $B(L[1, j], G)$ (a *set*) where $L[1, j] = (v_1, v_2, \ldots, v_j)$. `Boundary` must run in $O(j + m)$ in the worst-case.

Solution:

Algorithm 4 `Boundary` (G (graph), L (list), j (integer))

Input: A graph $G = (V, E)$, a list L, and an integer j
Output: The boundary B (set) of $L[1, j]$ with respect to G.

```
1: U ← L[1, j]
2: B ← ∅
3: for i ← 1 to j by increment of 1 do
4:     for all v that are neighbours of L[i] do
5:         if v ∉ U then
6:             Insert v in B
7:         end if
8:     end for
9: end for
10: Return B
```

Clit algorithm

Depth-first path and removing a vertex

Let $G = (V, E)$ with $V = \{1, 2, \ldots, n\}$ be a connected undirected graph and let $L = (v_1, v_2, \ldots, v_n)$ be a depth-first path starting at $v_1 \in V$. Let $\{x, y\} \in E$ be such that $x = v_i$ and $y = v_j$ for some $i < j$. We denote $G' = (V, E')$ the partial graph obtained from G by removing $\{x, y\}$. The goal of this exercise is to study depth-first paths on G'.

Question 9a: Consider $G = (V, E)$ with $V = \{1, 2, 3, 4, 5, 6, 7, 8\}$ and

$$E = \big\{\{1,2\}, \{1,6\}, \{2,3\}, \{2,8\}, \{3,4\}, \{4,5\}, \{4,6\}, \{6,7\}, \{7,8\}\big\}.$$

Let $L = (1, 2, 3, 4, 5, 6, 7, 8)$. Is L a depth-first path for G?

Solution: Yes. L is a valid depth-first path because starting from $L[1] = 1$, the vertex v_i for $i \in \{2, 3, 4, 5, 6, 7\}$ is adjacent to the last visited vertex from $(v_1, \ldots, v_i)$.

Question 9b: Let L as in (**3a**), $x = 4$, and $y = 6$. Is L a depth-first path of G'?

Solution: No.

Maximum weight spanning trees

A tree is a graph which is connected and has no cycles. A spanning tree of an undirected graph $G = (V, E)$ is a tree subgraph of G containing all nodes of the graph G. Assuming all edges $e \in E$ have positive weights c_e, we are interested in finding a maximum total weight spanning tree of G. Let the order $|V| = n$ and the size $|E| = m$.

Question 10: Formulate an integer optimization problem for this purpose.

Solution: Using the notation of the problem, the objective function $f : \{0, 1\}^m \to [0, \infty)$ is given

$$\sum_{e \in E} c_e x_e.$$

The goal is to maximize f therefore subject to the constraint:

$$\sum_{e \in E} x_e \leq |V| - 1 \quad \text{(order of a tree)}$$

A tree has no cycle. We must prevent cycle in the feasible set of solutions. For each cycle $C \subset V$, define $E(C)$ as the set of all edges of the cycle C. Then for all $C \subset V$ such that $|C| \geq 3$, the other constraint is

$$\sum_{e \in E(C)} x_e \leq |C| - 1 \quad \text{for all } C \subset V \text{ and } |C| \geq 3. \text{ (no cycle)}$$

Minimum weight spanning trees

Using the previous problem as a reference and assuming all edges $e \in E$ have a positive weights c_e, we are interested in finding a minimum total weight spanning tree of G.

Solution: There are two changes w.r.t. to previous problem: (1) change max for min, and (2) and the inequality for the order of the tree to an equality. If not (2), then minimization would simply yield an empty structure.

Question 11: Formulate an integer optimization problem for this purpose.

Steiner trees

Given a directed graph $G = (V, E)$ with edge weights c_{ij} for $(i, j) \in E$, a root vertex $r \in V$ (in-degree of r is 0), a subset of *destination* nodes $T \subset V \setminus \{r\}$. The Steiner tree problem consists in identifying a partial arborescence (an arborescence is a directed version of a tree graph, connected and acyclic) rooted at r and allowing it to reach vertex in T at minimal cost. The Steiner tree may also touch one or more vertices in $B(T \cup \{r\}, G)$. (B stands for the boundary) A typical application of this problem is in telecommunications where we wish to send a signal to some receivers passing by intermediate nodes if necessary.

Question 12: Formulate the Steiner tree problem as an integer optimization problem.

Solution: Let $x_{ij} \in \{0, 1\}$ be 0 is edge $(i, j) \in E$ is not selected and 1 if it is selected. The objective function is

$$\sum_{(i,j)\in E} c_{ij} x_{ij}.$$

The root must have in-degree 0 and all destination nodes must have in-degree 1. Hence, in terms of x_{ij}, we have already two set of constraints:

$$\sum_{(i,r)\in E} x_{ir} = 0,$$
$$\sum_{(i,t)\in E} x_{it} = 1 \quad \text{for all } t \in T.$$

Every cut that contains r and does not contain a terminal node should contain at least one selected arc otherwise it would be impossible to reach a destination $t \in T$ from the root r. For notational convenience, if U is a cut of V, then let us denote $A(U)$ the set of edges $(i, j) \in E$ such that $i \in U$ and $j \notin U$. Therefore, we have the following additional set of constraints:

$$\sum_{(i,j)\in A(U)} x_{ij} \geq \sum_{(i,t)\in E} x_{it} \quad \forall\, U \subset V \text{ such that } r \in U \text{ and } t \in V \setminus U.$$

Approximating solutions to the Travelling Salesperson Problem (TSP) on Euclidean graphs and Minimum Spanning Tree (MST)

Let $G = (V, E) = K_n$, the complete undirected with n vertices. Let $w : E \to \mathbb{N}$ be a weight function on K_n. (For simplicity, the weights are natural numbers, but they could be any positive real numbers for a reason that is given in the next sentence.) For $i, j \in V$, we write $w(\{i, j\} = c_{ij}$ and we may think of c_{ij} as the cost (or distance) of the edge $\{i, j\}$ for $1 \leq i, j \leq n$. We *assume* that the distances satisfy the triangle inequality, that is,

$$c_{ij} \leq c_{ik} + c_{kj} \quad \text{for all } i, j, k \in V.$$

We say that the graph G is Euclidean if it has a weight function that satisfies the triangle inequality.

We also call **clique of order** ℓ, a complete graph made of $\ell > 0$ vertices.

An **Euler circuit** of a graph is a cycle that contains all the edges exactly once. A **Hamiltonian circuit** of a graph is a cycle that contains all the vertices exactly once (except for first and last one).

Goal: The TSP problem (on an Euclidean graph for our purpose) consists of finding a Hamiltonian circuit of minimal length. In general, the problem is NP-complete, that is, there is *no* deterministic polynomial time algorithm that runs in time of the size of the graph, which can find an *exact* solution. (The size here is $n(n-1)/2$, which is the number of edges.) Nevertheless, we explain below how to find a *good* approximation in deterministic polynomial time by using MSTs.

If T is a MST of G, then we may write T_r for the subtree of T rooted at $r \in V$.

First, we start by considering the following algorithm:

Algorithm 5 `Euler` (v (vertex), T (MST of an Euclidean K_n))

1: $L \leftarrow v$ // L is a list.
2: **for all** child w of v in T_v **do**
3: $\quad L \leftarrow \texttt{Append}(L, \texttt{Append}(\texttt{Euler}(w, T_w), v))$
4: **end for**
5: **Return** L

NEXT PAGE

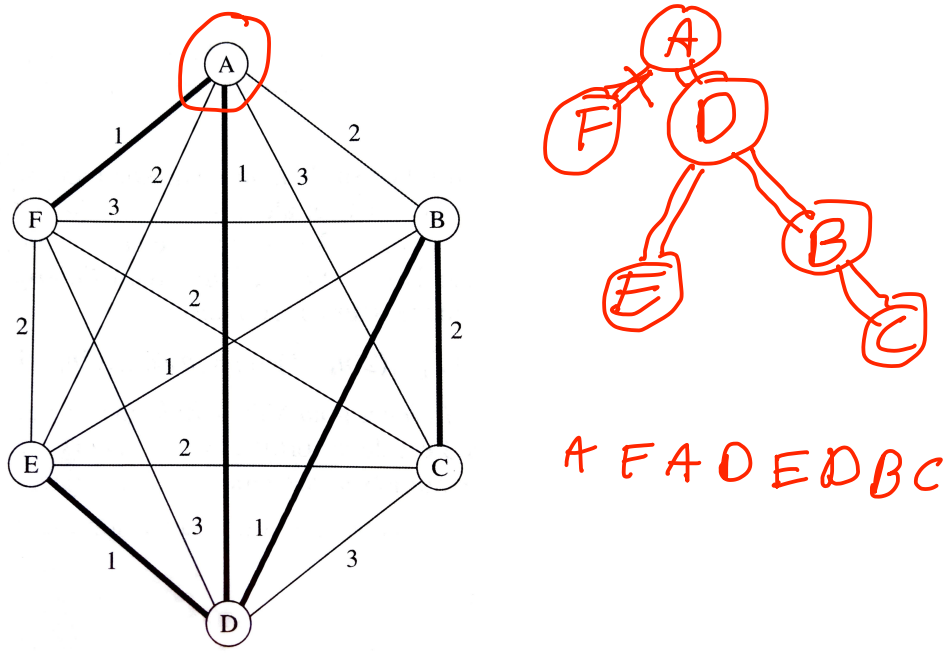


Figure 1: Example of an Euclidean K_6

Question 13a: What is the list returned by Algorithm `Euler`(A, H_A) on the case shown on Figure 1? The MST H_A is shown with bold edges.

Question 13b: What is the complexity of Algorithm `Euler`?

Let $(v_0, v_2, \ldots, v_q)$ be the returned list from `Euler`(v, T) for some MST T of $G = (V, E)$ and $v \in V$.

Question 13c1: Show that $q = 2(n-1)$.

Question 13c2: Show that for all $k \in \{0, 1, \ldots, q-1\}$, $\{v_k, v_{k+1}\}$ is an edge of T.

Question 13c3: Show that every edge of T appears twice in $(v_0, v_2, \ldots, v_q)$.

Question 13c4: Show that every vertex of T is listed at least once in $(v_0, v_2, \ldots, v_q)$.

We extract from $L = (v_0, v_2, \ldots, v_q)$ the longest sublist $L' = (v_{i_0}, v_{i_1}, \ldots, v_{i_p})$ *defined* as follow:

1. $i_0 = 0$
2. For all $k \in \{1, \ldots, p\}$, i_k is the smallest index such that v_{i_k} is not a vertex of $(v_{i_0}, v_{i_1}, \ldots, v_{i_{k-1}})$.

Question 13d: Find L' for L obtained from **7a**?

Question 13e: Show that L' is a Hamiltonian cycle of G and that $w(L') \leq 2w(T)$.

Let $w^\star$ be the minimum distance of a Hamiltonian cycle on G.

Question 13f: Show that $\frac{w(L')}{w^\star} \leq 2$.

Question 7a: Let $G = (V, E)$ be the undirected simple connected graph with

5

$V = \{a, b, c, d, e, f, g\}$ and matrix representation given by

$$\begin{matrix} A & b & C & D & \epsilon & F & G \end{matrix}$$

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \end{pmatrix}.$$

Row and columns indices follow the alphabetical order of the vertices. Let $L = \{a, g, c, b, f, d, e\}$ be a list. Explain why L represents a breadth-first path. First, the list L must contain all the vertices of G, and what else must L satisfy to represent a valid breadth-first path?

Solution: L is a valid breadth-first path because starting from a, the vertex v_i for $i \in \{2, 3, 4, 5, 6, 7\}$ is adjacent to the first visited vertex from $(v_1, \ldots, v_i)$.

Question 7b: What does `ALGO_1` compute on given input G and L from (**1a**)? (We have seen it in class.)

Solution: The breadth-first tree represented by L with respect to the graph G.

Question 7c: What does `ALGO_1` return on inputs G and L from (**1a**).

Solution: $\{(a, g), (a, c), (a, b), (g, f), (c, d), (b, e)\}$