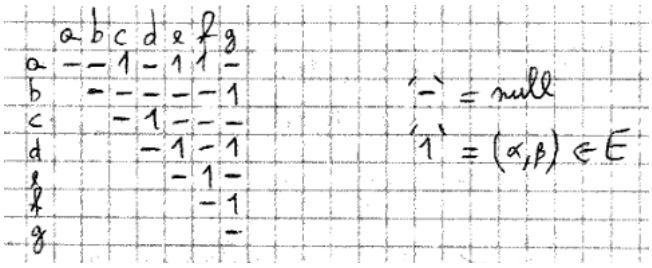
1) Given the following graph represented by the adjacency matrix:

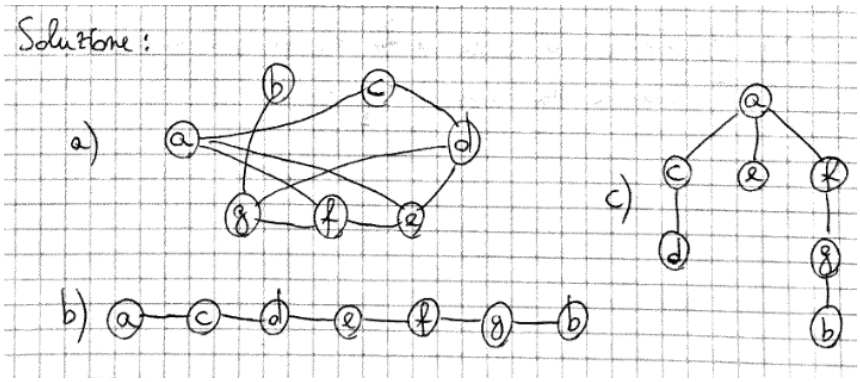


a) Draw the graph

b) Execute DFS from vertex a and show the obtained DFS tree

c) Same as previous point but for BFS

Solution



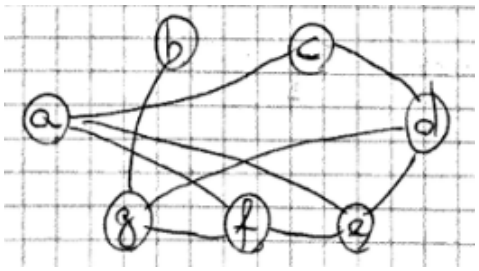
2) Given the following weighted graph, represented by an adjacency matrix:

a) list the MST edges in the order they were determined by Kruskal's algorithm

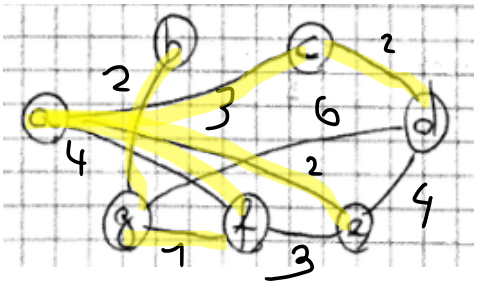
b) do the same using Prim's algorithm

Solution

Given:



We should consider for example weights on this graph in order to make it work, for example:



Kruskal: $\{g,f\},\{a,e\},\{a,c\},\{b,g\},\{c,d\},\{a,c\},\{a,f\}$

Prim:$\{a,c\},\{a,f\},\{a,e\},\{a,f\},\{b,g\},\{c,d\}$

Here of course in yellow, it appears the MST.

3) Describe an algorithm which, given a simple graph with 2 *vertices* s *and* t*, determines, if it exists, a path of minimal length between* s *and* t

This comes from the theory:

- $\forall v \in V$ add a field $L_V[v].parent$
- Modify $DFS(G,v)$ s.t. when a $DISCOVERY\ EDGE\ (v,w)$ is labeled
  - then $L_V[w].parent = v$ (v is parent of w in $DISCOVERY\ EDGE$ tree)
- Run $DFS(G,s)$. Check if t has been visited
  - NO: then return "No path"
  - YES: starting from t, follow the "parent" label, so as to build a path from t to s
- Complexity: $O(m_s)$ where m_s is the number of edges of s connected component

4) Knowing that a tree with n *nodes has* $m = n - 1$ *edges, show that a connected graph with* n *edges has* $m \geq n - 1$ *edges*

Sure, here's a concise version of the explanation written in plain text without LaTeX formatting, suitable for direct copying:

A tree is a connected graph without cycles, and one of its fundamental properties is that a tree with n nodes always has n-1 edges. This property is essential for ensuring that the graph remains connected without forming any cycles.

To extend this concept to general connected graphs, consider the following:

1. Definition and Tree Property: A tree with n nodes has exactly n-1 edges. This is the minimum number of edges required to connect all the nodes without forming cycles.

2. General Connected Graphs: If a connected graph has n nodes but fewer than n-1 edges, it cannot remain connected, which contradicts the definition of a connected graph. Therefore, a connected graph with n nodes must have at least n-1 edges.

3. Adding More Edges: Adding more edges to a tree (still with n nodes) introduces cycles but maintains connectivity. Hence, any connected graph with n nodes will have at least n-1 edges. More edges than this do not disrupt connectivity but simply add cycles.

4. Visualization Through Induction (Optional):

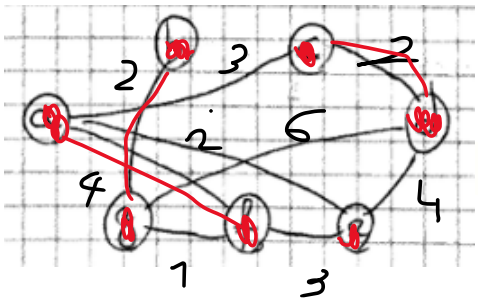
- Base Case: A single node (n=1) has 0 edges, aligning with n-1.

- Inductive Step: Assuming a graph with n nodes and n-1 edges is connected, adding one more node and at least one edge keeps it connected, leading to n nodes and n edges. The minimum n-1 edges ensure connectivity.

In conclusion, any connected graph with n nodes must have at least n-1 edges to ensure all nodes are connected. This minimum is achieved in tree structures, and adding more edges maintains or enhances connectivity by potentially introducing cycles.

5) *Given the following graph, show the set of edges returned by* $Approx\_Vertex\_Cover(G)$*, briefly describing the algorithm*.

Assuming once again we may be talking here about:



Remember logic and pseudocode of the approx algo:

- Choose *any* edge
- Add its endpoints to the solution
- "Remove" the covered edges
- Repeat

procedure $Approx\_Vertex\_Cover(G)$

$V' = \emptyset$

$E' = E$

while $E' \neq \emptyset$: do

$Let\ (u, v)\ be\ an\ arbitrary\ edge\ of\ E'$

$V' = V' \cup \{u, v\}$

$E' = E' \setminus \{(u, z), (v, w)\}$

$//\ remove\ edges\ that\ have\ u\ and\ v\ as\ endpoints$

return V'

Complexity: $O(n + m)$

So, here we would have the part in red above.

6)

(a) *Give matching and maximal matching definition*

(b) *Show a graph in which* $Approx\_Vertex\_Cover$ *returns a solution of cost exactly twice the optimal vertex cover*

A matching in a graph is a set of edges without common vertices, while a maximal matching is a matching which cannot be increased.

OPT: just one vertex

Consider the algorithm:

- The algorithm starts with an empty set V' (vertex cover set) and the original edge set E'. It iteratively selects an arbitrary edge (u, v) from E' and adds both vertices u and v to the vertex cover set V'. It then removes all edges from E' that are incident on either u or v
- The algorithm continues this process until E' becomes empty, meaning all edges have been covered by the selected vertices in V'. Finally, it returns V' as the approximate vertex cover

The key observation is that for each edge (u, v) selected, at least one of u or v must be present in the optimal vertex cover OPT. This is because OPT must cover all edges, and (u, v) is an edge in the original graph.

Therefore, during each iteration when an edge (u, v) is processed, the algorithm adds at most two vertices to V', while the optimal vertex cover OPT must contain at least one of these two vertices.

Consequently, we can establish the following inequality:

$$|V'| \leq 2 * |OPT|$$

The bound is tight; ensuring the greedy choice is 2 vertices and the optimal choice is just one vertex, we will have that $\frac{|V'|}{|OPT|} = 2 \leq 2$.

7) Show that if a graph has a minimum cut of cardinality t, then it has at least $\frac{tn}{2}$ edges

Solution: By definition of minimum cut we have $d(v) \geq t$ for any vertex $v \in V$, where $d(v)$ denotes the degree of v. Then, summing up over all the n vertices we obtain $\sum_{v \in V} d(v) \geq tn$. The claim follows by observing that $\sum_{v \in V} d(v) = 2m$.