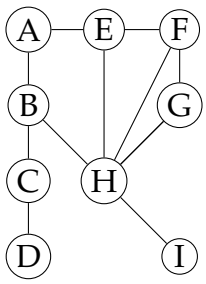
Warm-up

Problem 1. Consider the following undirected graph.



(a) A
B E
C H F
D I G

(b) A B C D H I G F E

a) Starting from A, give the layers the breadth-first search algorithm finds.

b) Starting from A, give the order in which the depth-first search algorithm visits the vertices.

Problem solving

Problem 2. An undirected graph $G = (V, E)$ is said to be bipartite if its vertex set V can be partitioned into two sets A and B such that $E \subseteq A \times B$. Design an $O(n + m)$ algorithm to test if a given input graph is bipartite using the following guide:

a) Suppose we run BFS from some vertex $s \in V$ and obtain layers $L_1, \ldots, L_k$. Let (u, v) be some edge in E. Show that if $u \in L_i$ and $v \in L_j$ then $|i - j| \leq 1$.

b) Suppose we run BFS on G. Show that if there is an edge (u, v) such that u and v belong to the same layer then the graph is not bipartite.

c) Suppose G is connected and we run BFS. Show that if there are no intra-layer edges then the graph is bipartite.

d) Put together all the above to design an $O(n + m)$ time algorithm for testing bipartiness.

Problem 3. Give an $O(n)$ time algorithm to detect whether a given undirected graph contains a cycle. If the answer is yes, the algorithm should produce a cycle. (Assume adjacency list representation.)

Problem 4. Let $G = (V, E)$ be an n vertex graph. Let s and t be two vertices. Argue that if $\text{dist}(s, t) > n/2$ then there there exists a vertex $u \neq s, t$ such that every path from s to t goes through u.

Problem 5. In a directed graph, a *get-stuck* vertex has in-degree $n - 1$ and out-degree 0. Assume the adjacency matrix representation is used. Design an $O(n)$ time algorithm to test if a given graph has a get-stuck vertex. Yes, this problem can be solved without looking at the entire input matrix.

Problem 6. Let G be an undirected graph with vertices numbered $1 \dots n$. For a vertex i define $\text{small}(i) = \min\{j : j \text{ is reachable from } i\}$, that is, the smallest vertex reachable from i. Design an $O(n + m)$ time algorithm that computes $\text{small}(i)$ for *every* vertex in the graph.

Problem 7. In a connected undirected graph $G = (V, E)$, a vertex $u \in V$ is said to be a cut vertex if its removal disconnects G; namely, $G[V - u]$ is not connected.

The aim of this problem is to adapt the algorithm for cut edges from the lecture, to handle cut vertices.

a) Derive a criterion for identifying cut vertices that is based on the `down-and-up`$[\cdot]$ values defined in the lecture.

b) Use this criterion to develop an $O(n + m)$ time algorithm for identifying all cut vertices.

Problem 8. Let T be a rooted tree. For each vertex $u \in T$ we use T_u to denote the subtree of T made up by u and all its descendants. Assume each vertex $u \in T$ has a value $A[u]$ associated with it. Let $B[u] = \min\{A[v] : v \in T_u\}$. Design an $O(n)$ time algorithm that given A, computes B.

Problem 9. Let G be a connected undirected graph. Design a linear time algorithm for finding all cut edges by using the following guide:

a) Derive a criterion for identifying cut edges that is based on the `down-and-up`$[\cdot]$ values defined in the lecture.

b) Use this criterion to develop an $O(n + m)$ time algorithm for identifying all cut edges.

Problem 2. An undirected graph $G = (V, E)$ is said to be bipartite if its vertex set V can be partitioned into two sets A and B such that $E \subseteq A \times B$. Design an $O(n + m)$ algorithm to test if a given input graph is bipartite using the following guide:

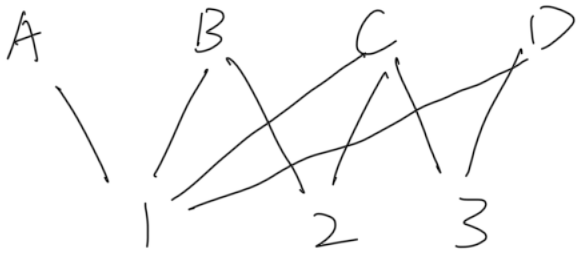
a) Suppose we run BFS from some vertex $s \in V$ and obtain layers $L_1, \ldots, L_k$. Let (u, v) be some edge in E. Show that if $u \in L_i$ and $v \in L_j$ then $|i - j| \leq 1$.

b) Suppose we run BFS on G. Show that if there is an edge (u, v) such that u and v belong to the same layer then the graph is not bipartite.

c) Suppose G is connected and we run BFS. Show that if there are no intra-layer edges then the graph is bipartite.

d) Put together all the above to design an $O(n + m)$ time algorithm for testing bipartiness.

Example of Bipartite



即数字 set 和字母 set 之间的 vertex 才有 Edge

而同 set 之间没有 edge

(a) 因为 Edge 必须连接 1 个字母和 1 个数字，

而 layer 又因为数字和字母分层 即

A
1
B C D
2 3

所以 i 和 j 之间的差一定小于 1

Problem 3. Give an $O(n)$ time algorithm to detect whether a given undirected graph contains a cycle. If the answer is yes, the algorithm should produce a cycle. (Assume adjacency list representation.)

- Using DFS, if we find a back edge, then it means we find a cycle
- Up until the time we find the back edge, we have only discovered tree edge that's upper bound by $n-1$
- Thus the overall running time is $O(n)$

Problem 4. Let $G = (V, E)$ be an n vertex graph. Let s and t be two vertices. Argue that if $\text{dist}(s, t) > n/2$ then there there exists a vertex $u \neq s, t$ such that every path from s to t goes through u.

- Proof by contradiction
- BFS begin from S

Assume: Each layer need to has at least 2 vertex; otherwise, it satisfy the requirement of u.

- Between S and t, there are $k-1$ layers

l_0 l_k

l_0 0 ... $k=3$

$3-1=2$

- $2 \times (k-1) = k-2$ means among them, there are $k-2$ nodes

Since $k = \text{dist}(S, t) > \frac{n}{2}$, we have $n-2$ nodes more than

- (see "...")
- Contradiction: say if we have $n-1$, then the total # of node is $n-1+s+t = n-1+2 = n+1$

Problem 5. In a directed graph, a *get-stuck* vertex has in-degree $n-1$ and out-degree 0. Assume the adjacency matrix representation is used. Design an $O(n)$ time algorithm to test if a given graph has a get-stuck vertex. Yes, this problem can be solved without looking at the entire input matrix.

- if i is a get-stuck vertex, then { row i is all 0's; col i is all 1s except for (i,i)
- Start from top left corner ~~to~~ right
  ① if at (i,j) entry, skip to its right entry
  ② if at (i,j) for $i \neq j$, then { move down if we see a 1; move right if we see a 0
- The algorithm end when we exit the matrix
  Case1: exit bottom boundary, no set-stuck vertex and takes $O(n)$

Case2: exit right boundary: we need to do one more check. Because consider
0 1 0
1 0 0
0 0 1
it will check both i row and i col to determine whether i is a get-stuck vertex which takes
$O(n) + O(n-1) + O(n-1)$
不包含 diagonal entry

$n=4$

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

每个 Entry 表示: (u,v) $(u \to v)$, 1表示存在, 0表示不存在

Problem 6. Let G be an undirected graph with vertices numbered $1 \ldots n$. For a vertex i define small$(i) = \min\{j : j \text{ is reachable from } i\}$, that is, the smallest vertex reachable from i. Design an $O(n+m)$ time algorithm that computes small(i) for *every* vertex in the graph.

use DFS

Problem 7. In a connected undirected graph $G = (V, E)$, a vertex $u \in V$ is said to be a cut vertex if its removal disconnects G; namely, $G[V - u]$ is not connected.

The aim of this problem is to adapt the algorithm for cut edges from the lecture, to handle cut vertices.

a) Derive a criterion for identifying cut vertices that is based on the `down-and-up`$[\cdot]$ values defined in the lecture.

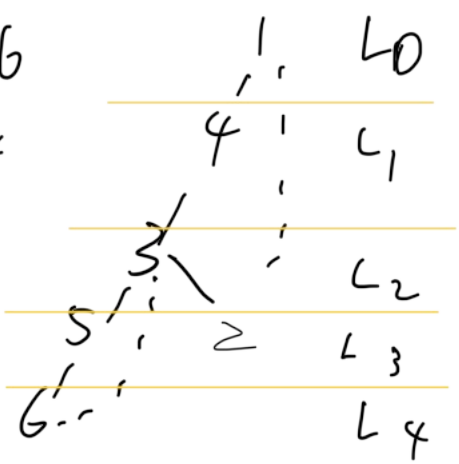
b) Use this criterion to develop an $O(n+m)$ time algorithm for identifying all cut vertices.

1. Build a DFS Tree, record level of each vertex

2. DFS Edges down 表示通过 任意一个 DFS-Edges 尽可能下降

One Back Edge up 表示通过 一个 Back Edge 回到的层数 最终目的是返回尽可能高的 layer

	1	2	3	4	5	6
level	0	3	2	1	3	4
down_and_up	0	0	0	0	2	2



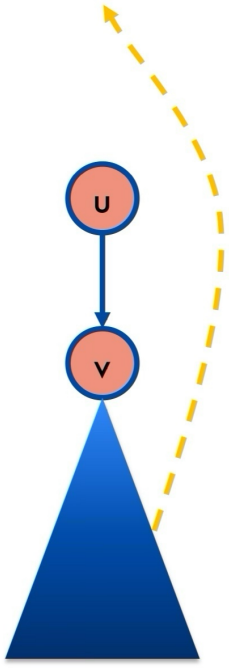
Compute a DFS tree of the input graph G=(V, E)

For every u in V, compute level[u], its level in the DFS tree

For every vertex v compute the highest level that we can reach by taking DFS edges down the tree and then one back edge up. Call this down_and_up[v]

Fact: A DFS edge (u, v) where u = parent[v] is not a cut edge if and only if down_and_up[v] ≤ level[u]

Basis of an O(n+m) time algorithm for finding cut edges



Problem 8. Let T be a rooted tree. For each vertex $u \in T$ we use T_u to denote the subtree of T made up by u and all its descendants. Assume each vertex $u \in T$ has a value $A[u]$ associated with it. Let $B[u] = \min\{A[v] : v \in T_u\}$. Design an $O(n)$ time algorithm that given A, computes B.

- We use Post-order to traverse the tree
- for each subtree, we need to calculate its B

that's:

$$B[u] = \min_{x \in T_u} A[u] = \min\left\{A[u], \min_{x \in T_{v_1}} A[x], \cdots, \min_{x \in T_{v_k}} A[x]\right\}$$

$$= \min\{A[u], B[v_1], \cdots, B[v_k]\}$$

where $v_1, \cdots, v_k$ are children of u

- 树中 edge 的数量是 $(n-1)$.
- 而上面提到的要对所有 vertices 用，即要从 parent ⇒ children 再从 children 回到 parents，因此为 $2(n-1)$

Problem 9. Let G be a connected undirected graph. Design a linear time algorithm for finding all cut edges by using the following guide:

a) Derive a criterion for identifying cut edges that is based on the `down-and-up`$[\cdot]$ values defined in the lecture.

b) Use this criterion to develop an $O(n+m)$ time algorithm for identifying all cut edges.

(a) · get a DFS-tree,

· only DFS-tree edge can be cut edge

· let u be v's parent

if Down_AND_UP(v) > level(u), then edge (u,v) is not cut edge,