Free University of Bozen-Bolzano – Faculty of Computer Science
Master in Computational Data Science
Algorithms for Data Processing – A.Y. 2018/19
Final Exam – 11/Feb/2019

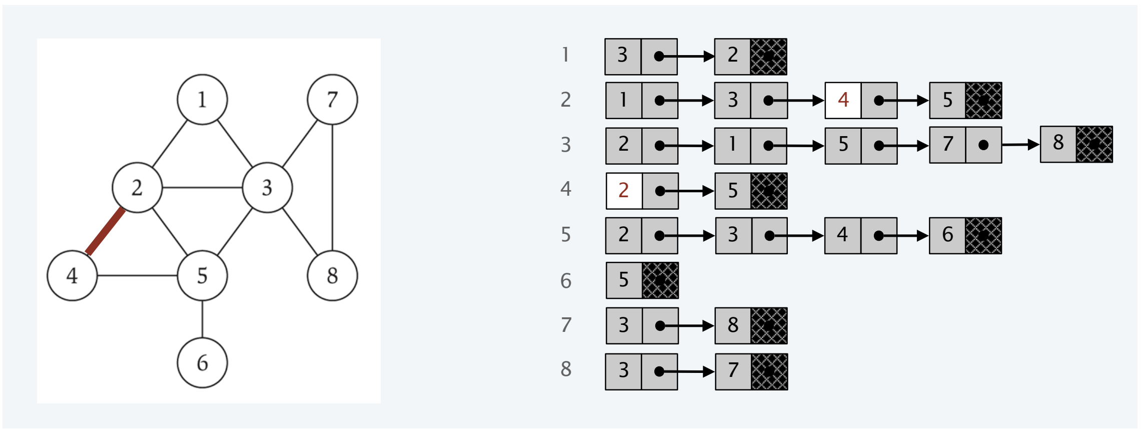
Alessandro Artale

`Time: 3`^{h}

`This is a closed book exam: the only resources allowed are blank papers, pens, and your head. Explain your reasoning.`
`Write clearly, in the sense of logic, language and legibility. The clarity of your explanations affects your grade.`
The use of Pencils is not allowed!

Note: Total points – 38.
Points to reach 30cumLaude: 32

Problem 1 [5 points] **BFS.**
Consider the following Graph with its associated Adjacency List:



and the following BFS Algorithm:

```
BFS(G,s)
    Discovered[s]=true;
    Discovered[u]=false, for all other u $\in V$;
    L[0]=s; layer counter i=0; spanning tree T=s;
    While L[i]$\neq\emptyset$
      Initialize an empty list L[i+1]
      For each node $u \in L[i]$
        For each edge (u,v) incident to u;
          If Discovered[v]=false then
            Discovered[v]=true;
            Add edge (u,v) to tree T;
            Add v to the list L[i+1]
          EndIf
        EndFor
      EndFor
      i=i+1;
    Endwhile
```

Answer the following questions:

(a) Show the spanning tree generated by the BFS Algorithm when calling the Algorithm with starting node $s = 3$. [3 Points]

(b) Show the BFS Complexity and give an exhaustive explanation. [2 Points]

Problem 2 [5 points] **Bipartite Graphs.**

(a) Write a Pseudo-Code that is able to check whether a Graph is a Bipartite Graph. [3 POINTS] In particular:

- Assume that the input Graph is connected;
- Reuse the BFS algorithm of the previous exercise;
- Show just the part of the BFS algorithm that changes (i.e., it is not needed to show the Algorithm in full, also the spanning tree can be avoided).

(b) Prove that the Algorithm you provided is correct. Give an exhaustive explanation. [2 POINTS]

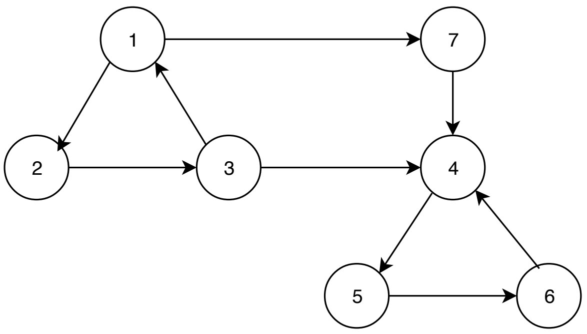
Problem 3 [5 points] **Tarjan Algorithm.**

Given the Tarjan Algorithm:

```
Strong-Components(G)
index=1; v.index=0, for all $v \in V$;            /* index set to 1 and vertex index set to 0 */
S = [];                                           /* The stack is initialized empty */
for each $v \in V$                                /* depth-first search for each vertex */ do
    if v.index=0 then Tarjan(v)                   /* that was not already visited */

Tarjan(v);
v.index = index; v.min-index = index;             /* set vertex indices to the current index */
index = index + 1; S.push(v);
for each edge (v, v') incident to v               /* checks all vertices adjacent to v */ do
    if v'.index=0 then                            /* vertex not already visited */
        Tarjan($v'$);                             /* DFS Recursion */
        v.min-index = min(v.min-index, v'.min-index)
    else if v' is inside S then
        v.min-index = min(v.min-index, v'.min-index);   /* $v'$ has a path to v */
if v.min-index = v.index;         /* v is a representative vertex and an SCC has been found */
 then
    repeat
        v' = S.pop();
        output v';                                /* output SCC */
    until (v' != v);
```

and the following input Graph:



(a) Produce an Adjacency List for the Input Graph. [1 POINT]

(b) Using the above constructed Adjacency List, emulate the running of the Algorithm by showing: [4 POINTS]

- Nesting of recursive calls;
- Values of the variables and of the stack at each recursive call;
- Intermediate output if a strongly connected component is discovered.

Problem 4 [9 points] **Ford-Fulkerson Algorithm.**

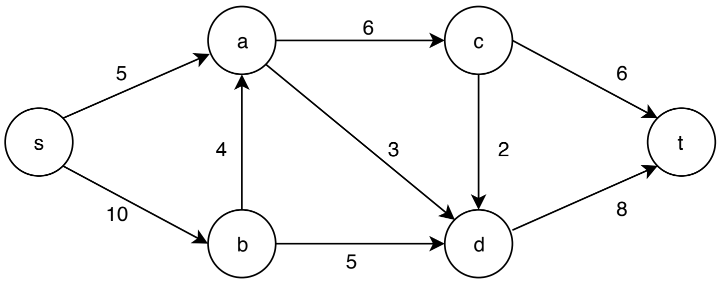
Given the Ford-Fulkerson Algorithm:

```
MAX-FLOW(G)
$f(e) = 0$ for all $e \in G$;
$G_0 = G$;                                          /* Initialize the residual graph $G_0$ */
while there is an s-t path, $P$, in the residual graph $G_f$ do
    if $P$ is a simple s-t path in $G_f$ then
        $f' =$ AUGMENT($f, P$);
        $G_{f'} =$ UPDATE($G_f$,P,$f'$);                        /* Update $G_f$ to $G_{f'}$ */
        $f \leftarrow f'$;
return $f$

AUGMENT(f,P)
b = BOTTLENECK(P, f);
for each edge $(u, v) \in P$ do
    if $e = (u, v)$ is a forward edge then
        $f(e) = f(e) + b$ in $G$
    else
        $f(e) = f(e) - b$ in $G$;                   /* $(u, v)$ is a backward edge, and $e = (v, u)$ */
return $f$

UPDATE($G_f$,P,f')
compute $b$ from $f$ and $f'$;
for each $(u, v)$ in $P$ do
    c(u,v) = c(u,v) - b;
    if c(u,v) =0 then
        delete edge $(u, v)$ from $G_f$
    if $(v, u) \notin G_f$ then
        add edge $(v, u)$ to $G_f$
    c(v,u) = c(v,u) + b;
return $G_f$
```

and the following input Graph:



(a) Run the Ford-Fulkerson Algorithm showing at each step: [5 Points]

- The Residual Graph;
- The Augmenting Path, and
- The Bottleneck value of the Augmenting Path.

(b) A min-cut and the corresponding flow across the cut. [1 Point]

(c) Prove the Correctness of the Ford-Fulkerson Algorithm. In particular, prove the following Theorem: [3 Points]
Theorem. If f is an s-t flow such that there is no s-t path in the residual graph G_f (Augmenting Path) then there is an s-t cut (A, B) in G for which $val(f) = c(A, B)$.

Problem 5 [7 points] **Net-Flow.**

You are given the following problem that you should solve using Net-Flow techniques. The problem is about scheduling the working days of Doctors in a Hospital.

`INPUT`

- The Hospital wants to schedule the Doctors for the nest "n" Days;
- The Hospital requires *exactly* "p_i" Doctors to be present on each Day $i = 1, \dots, n$;
- There are "k" Doctors;
- Each Doctor has a set of Available Days, A_i, for $i = 1, \dots, k$;

`OUTPUT`

- A set of Working Days for each Doctor, W_i, for $i = 1, \dots, k$ such that:
  - $W_i \subseteq A_i$, for each $i = 1, \dots, k$;
  - *Exactly* "p_i" Doctors works on each Day $i = 1, \dots, n$.

(a) Explain in details how to encode the problem as a Net-Flow problem. [5 Points]

(b) Discuss the generalization of Net-Flow extended with Circulation with Supplies and Demands. [2 Points]

Problem 6 [7 points] **NP Problems.**

Given the following two NP problems:

Definition of `Vertex-Cover`. Given a graph $G = (V, E)$ and an integer k, is there a subset of k (or fewer) vertices such that each edge is incident to at least one vertex in the subset?

Definition of `Independent-Set`. Given a graph $G = (V, E)$ and an integer k, is there a subset of k (or more) vertices such that no two are adjacent?

(a) Shows that `Independent-Set` $\leq_P$ `Vertex-Cover`. [3 Points]

(b) Give the Definition of a problem being NP-complete. [2 Points]

(c) Discuss the notion *Degrees of Approximability* when applied to NP problems and say what is the Degrees of Approximability for the Knapsack problem. [2 Points]