Pset 3

Darwin Do

February 22, 2022

(a) Darwin Do

(b) 919941748

(c) Collaborators:

(d) I have followed the academic integrity and collaboration policy

(e) Hours:

1 Checking Connectivity

(a)

```
function RedFind(G, u, v)
    for all v ∈ V do
        visited[v] ← False
    Q ← MakeQueue()                          ▷ standard FIFO queue
    Q.push(u)
    while Q not empty do
        q ← Q.pop()
        if q == v then return True
        visited[q] ← True
        for all (q, v) ∈ E do
            if color(q, v) == RED and not visited[v] then
                Q.push(v)
    return False
```

Correctness
Blah blah blah

Running Time

(b) **function** DFS(G, u, v)

```
function DFS(G, u, v)
    for all w ∈ V do
        visited[w] ← False
    visited[u] ← True
    for all (u, w) ∈ E do
        if COLOR(u, w) == Red then
            if EXPLORE(G, w, v, Red) then return True
        else if EXPLORE(G, w, v, Blue) then return True
    return False

function EXPLORE(G, w, v, edgeColor)
    visited[w] ← True
    if w == v then return True
    for all (w, u) ∈ E do
        if edgeColor == Blue then
            if COLOR(w, u) == Blue and not visited[u] then
                if EXPLORE(G, u, v, Blue) then return True
        else                                    ▷ previous edge is still on red path
            if not visited[u] then
                if EXPLORE(G, u, v, COLOR(w, u)) then return True
    return False
```

2 Road Trip

(a)
```
function IsReachable(G, s, t)
    for all v ∈ V do
        visited[v] ← False
    Q ← MakeQueue()                          ▷ standard FIFO queue
    Q.push(s)
    while Q not empty do
        q ← Q.pop()
        if q == t then return True
        visited[q] ← True
        for all (q, v) ∈ E do
            if L ≥ ℓ(v, q) and not visited[v] then
                Q.push(v)
    return False
```

(b)
```
function LowestGas(G, ℓ, s, t)
    for all v ∈ V do
        maxL[v] ← ∞
    maxL[s] ← 0
    P ← MakeQueue(V)     ▷ priority queue with maxL values as keys
    while P not empty do
        v ← ExtractMin(P)
        for all (v, w) ∈ E do
            if maxL[w] > maxL[v] and maxL[w] > ℓ(v, w) then
                maxL[w] ←max(maxL[v], ℓ(v, w))
                ChangeKey(P, w)
    return maxL[t]
```

3 Counting Shortest Paths

Struct Definition

```
struct {
    int dist, numPaths;
} PathStruct;
```

function NumShortest(G, ℓ, s, t)
  for all $v \in V$ **do**
    paths[v].dist $\leftarrow \infty$
    paths[v].numPaths $\leftarrow 0$
  paths[s].dist $\leftarrow 0$
  paths[s].numPaths $\leftarrow 1$
  $P \leftarrow$ MakeQueue(V) ▷ priority queue with dist values as keys
  while P **not** empty **do**
    $v \leftarrow$ ExtractMin(P)
    for all $(v, w) \in E$ **do**
      if paths[w].dist $>$ paths[v].dist $+ \ell(v, w)$ **then**
        paths[w].dist $\leftarrow$ paths[v].dist $+ \ell(v, w)$
        paths[w].numPaths $\leftarrow$ paths[v].numPaths
      else if paths[w].dist $==$ paths[v].dist $+ \ell(v, w)$ **then**
        paths[w].numPaths $\leftarrow$ paths[w].numPaths $+$ paths[v].numPaths
  return paths[t].numPaths

4 Spanning Tree with Leaves

We implement Kruskal's algorithm two times.

function LEAFSPANNINGTREE(G, U, w)
    for all $v \in V$ **do**
        MAKESET(v)
    $F \leftarrow \{\ \}$
    sort edges E by increasing weight
    $E' \leftarrow \{\{u, v\} \in E | u \notin U \wedge v \notin U\}$
    $E'' \leftarrow \{\{u, v\} \in E | u \in U \oplus v \in U\}$
    for all $\{u, v\} \in E'$ **do**
        if FIND(u) $\neq Find(v)$ **then**
            $F \leftarrow F \cup \{\{u, v\}\}$
            UNION($\{u, v\}$)
    for all $\{u, v\} \in E''$ **do**
        if FIND(u) $\neq Find(v)$ **then**
            $F \leftarrow F \cup \{\{u, v\}\}$
            UNION($\{u, v\}$)

5 Perfect Matching in a Tree

```
function CheckPerfectMatching(G)
    Q ← MakeQueue()                                      ▷ standard FIFO queue
    P ← MakeQueue()                                      ▷ standard FIFO queue
    for all v ∈ V do
        isPaired[v] ← False
        if degree of v == 1 then
            Q.push(v)
    while Q not empty and P not empty do
        while Q not empty do
            q ← Q.pop()
            if isPaired[q] then
                continue
            r ← None
            for all {q, v} ∈ E do            ▷ at most 1 unpaired neighbor exists
                if not isPaired[v] then
                    r ← v
                    break
            if r == None then return False
            isPaired[q] ← True
            isPaired[r] ← True
            rNumUnpairedNeighbors ← 0
            rUnpairedNeighbor ← None
            for all {r, v} ∈ E do
                rNumUnpairedNeighbors ← rNumUnpairedNeighbors + 1
                rUnpairedNeighbor ← v
            if rNumUnpairedNeighbors == 1 then
                Q.push(rUnpairedNeighbor)
            else
                P.push(r)
        while P not empty do
            p ← P.pop()
            if isPaired[p] then
                continue
            numUnpairedNeighbors ← 0
            for all {p, v} ∈ E do
                if not isPaired[v] then
                    numUnpairedNeighbors ← numUnpairedNeighbors + 1
                if numUnpairedNeighbors > 1 then
                    skip to next p in queue
            Q.push(p)                          ▷ node only has 1 unpaired neighbor
    return True
```