CS5800: Algorithms — Spring '21 — Virgil Pavlu

Homework 12
Submit via Gradescope

Name: Shri Datta Madhira (NUID: 001557772)
Collaborators:

Instructions:

- Make sure to put your name on the first page. If you are using the LaTeX template we provided, then you can make sure it appears by filling in the `yourname` command.
- Please review the grading policy outlined in the course information page.
- You must also write down with whom you worked on the assignment. If this changes from problem to problem, then you should write down this information separately with each problem.
- Problem numbers (like Exercise 3.1-1) are corresponding to CLRS 3^{rd} edition. While the 2^{nd} edition has similar problems with similar numbers, the actual exercises and their solutions are different, so make sure you are using the 3^{rd} edition.

1. ***(20 points)*** *Exercise 26.1-3.*

Solution:

If the vertex u is on no path from s to t, there are two possibilities. Either there is no path from s to u, or/and from u to t.

Let us take the second condition that there is no path from u to t. Let us also assume that there is a vertex v ∈ V, such that f(u,v) > 0. If the flow is greater than zero that means a path exists from u to v. But v has a path to t since, u is the only odd vertex out of the bunch. That means, if we keep extending the path from v onwards, at some point we reach t. That means, we traversed a path from u to t which contradicts the argument. Therefore, the flow from u to any vertex v ∈ V must be 0, i.e., f(u,v) = f(v,u) = 0, to satisfy the condition that there is no path between u and t.

The same argument goes if we assume that there is no path between s to u.

Therefore, $f(u,v) = f(v,u) = 0$.

2. ***(20 points)*** *Exercise 26.1-4.*

Solution:

For f_1 and f_2 to be considered as flows, they need to satisfy the "Capacity constraint" and "Flow Conservation" properties.

Capacity Constraint:

$$0 \le \alpha f_1(u,v) + (1-\alpha)f_2(u,v) \le \alpha c(u,v) + (1-\alpha)c(u,v)$$

$$\Rightarrow 0 \le \alpha f_1(u,v) + (1-\alpha)f_2(u,v) \le \alpha c(u,v) + c(u,v) - \alpha c(u,v)$$

$$\Rightarrow 0 \le \alpha f_1(u,v) + (1-\alpha)f_2(u,v) \le c(u,v)$$

Therefore, the Capacity Constraint condition has been satisfied.

Flow Conservation:

$$\sum_{v\in V} \alpha f_1(u,v) + (1-\alpha)f_2(u,v)$$

$$\Rightarrow \alpha \sum_{v\in V} f_1(u,v) + (1-\alpha)\sum_{v\in V} f_2(u,v)$$

Since f1 and f2 are flows, $\sum_{v\in V} f_1(u,v) = \sum_{v\in V} f_1(v,u)$ and $\sum_{v\in V} f_2(u,v) = \sum_{v\in V} f_2(v,u)$. Substituting this in the above equation we get,

$$\Rightarrow \alpha \sum_{v\in V} f_1(v,u) + (1-\alpha)\sum_{v\in V} f_2(v,u)$$

$$\Rightarrow \sum_{v\in V} \alpha f_1(v,u) + (1-\alpha)f_2(v,u)$$

Therefore, the Flow Conservation property is also satisfied. Which means, $\alpha f_1(u,v) + (1-\alpha)f_2(u,v)$ is also a flow.

Therefore, the flows in a network form a ***convex set***.

3. ***(20 points)*** *Exercise 26.2-2.*

Solution:

The cut given in the question is $(\{s, v_2, v_4\}, \{v_1, v_3, t\})$.

The flow for this cut will be,

$$f(s,t) = f(s,v_1) + f(v_2,v_1) + f(v_4,v_3) + f(v_4,t) - f(v_3,v_2)$$

$$f(s,t) = 11 + 1 + 7 + 4 - 4$$

$$\boldsymbol{f(s,t) = 19}$$

The capacity of this cut will be,

$$c(s,t) = c(s,v_1) + c(v_2,v_1) + c(v_4,v_3) + c(v_4,t)$$

$$c(s,t) = 16 + 4 + 7 + 4$$

$$\boldsymbol{c(s,t) = 31}$$

4. ***(Extra Credit)*** *Exercise 26.2-10.*

Solution:

5. ***(30 points)*** *Implement Push-Relabel for finding maximum flow. Extra Credit: use relabel-to-front idea from Chapter 26.5 with the Discharge procedure.*

Solution:

```
#include <iostream>
#include <vector>

using namespace std;

struct Edge
{
    int flow, capacity, u, v;
    Edge(int flow, int capacity, int u, int v)
    {
        this->flow = flow;
        this->capacity = capacity;
        this->u = u;
        this->v = v;
    }
};
struct Vertex
{
    int height, excess;
    Vertex(int h, int e)
    {
        this->height = h;
        this->excess = e;
    }
};

class Graph
{
    int V;
    vector<Vertex> vertexes;
    vector<Edge> edges;

    bool push(int u);
    void relabel(int u);
    void preflow(int s);
    void updateReverseEdgeFlow(int i, int flow);
public:
    Graph(int V);
    void addEdge(int u, int v, int w);
    int getMaxFlow(int s, int t);
};
```

```
Graph::Graph(int V)
{
    this->V = V;
    for (int i = 0; i < V; i++)
        vertexes.push_back(Vertex(0, 0));
}

void Graph::addEdge(int u, int v, int capacity)
{
    edges.push_back(Edge(0, capacity, u, v));
}
int vertex_with_excess(vector<Vertex>& ver)
{
    for (int i = 1; i < ver.size() - 1; i++)
       if (ver[i].excess > 0)
            return i;
    return -1;
}

void Graph::preflow(int s)
{
    vertexes[s].height = int(vertexes.size());
    for (int i = 0; i < edges.size(); i++)
    {
        if (edges[i].u == s)
        {
            edges[i].flow = edges[i].capacity;
            vertexes[edges[i].v].excess += edges[i].flow;
            edges.push_back(Edge(-edges[i].flow, 0, edges[i].v, s));
        }
    }
}
void Graph::updateReverseEdgeFlow(int i, int flow)
{
    int u = edges[i].v, v = edges[i].u;
    for (int j = 0; j < edges.size(); j++)
    {
        if (edges[j].v == v && edges[j].u == u)
        {
            edges[j].flow -= flow;
            return;
        }
    }
    edges.push_back(Edge(0, flow, u, v));
}
```

```
bool Graph::push(int u)
{
    for (int i = 0; i < edges.size(); i++)
    {
        if (edges[i].u == u)
        {
            if (edges[i].flow == edges[i].capacity)
                continue;
            if (vertexes[u].height > vertexes[edges[i].v].height)
            {
                int flow = min(edges[i].capacity - edges[i].flow,
                   vertexes[u].excess);
                vertexes[u].excess -= flow;
                vertexes[edges[i].v].excess += flow;
                edges[i].flow += flow;
                updateReverseEdgeFlow(i, flow);
                return true;
            }
        }
    }
    return false;
}
void Graph::relabel(int u)
{
    int mh = INT_MAX;
    for (int i = 0; i < edges.size(); i++)
    {
        if (edges[i].u == u)
        {
            if (edges[i].flow == edges[i].capacity)
                continue;
            if (vertexes[edges[i].v].height < mh)
            {
                mh = vertexes[edges[i].v].height;
                vertexes[u].height = mh + 1;
            }
        }
    }
}
int Graph::getMaxFlow(int s, int t)
{
    preflow(s);
    while (vertex_with_excess(vertexes) != -1)
    {
        int u = vertex_with_excess(vertexes);
        if (!push(u))
            relabel(u);
    }
    return vertexes.back().excess;
}

int main(int argc, const char * argv[]) {
    int V = 6;
    Graph g(V);

    g.addEdge(0, 1, 16);
    g.addEdge(0, 2, 13);
    g.addEdge(1, 2, 10);
    g.addEdge(2, 1, 4);
    g.addEdge(1, 3, 12);
    g.addEdge(2, 4, 14);
    g.addEdge(3, 2, 9);
    g.addEdge(1, 4, 15);
    g.addEdge(3, 5, 20);
    g.addEdge(4, 3, 7);
    g.addEdge(4, 5, 4);

    int s = 0, t = 5;

    cout << "Maximum flow: " << g.getMaxFlow(s, t) << endl;
    return 0;
}
```

6. ***(15 points)*** *Explain in a brief paragraph the following sentence from textbook page 737: "To make the preflow a legal flow, the algorithm then sends the excess collected in the reservoirs of overflowing vertices back to the source by continuing to relabel vertices to above the fixed height $|V|$ of the source".*

Solution:

In push relabel algorithm we will relax the flow conservation property. we will say that inflow(v) ≥ outflow(v). Because of this relaxation, in preflow, we will have vertices that have excess. We will relabel them and push them according to the algorithm already provided.

At some point during this operation, we will arrive at a situation where we have saturated all the possible paths that are going into the sink from that particular vertex v, but it still has some excess > 0. These kind of excess vertices are then relabeled to satisfy the height constraint and the excess is then sent back to the source vertex where it disappears.

This operation then makes the preflow we have a valid flow by making these vertices obey the conservation of flow property.

7. ***(Extra Credit)*** *Exercise 26.4.4.*

Solution: