Network flow

Giulia Carocari, Giacomo Fabris, Francesco Lotito

Università degli Studi di Trento

ICPC Training @ UniTN
Day 7 - May 2, 2019

Table of Contents

Table of Contents

A - Inflation

Problem statement

Balloon capacity goes from 1 to n, you are given the amount of helium in each of the n helium canisters. Your goal is to assign gas canisters to balloons in a way such that the least full balloon still contains the maximum possible fraction of helium inside. If a ballon is filled beyond its capacity, it will explode.

A - Inflation

Problem statement

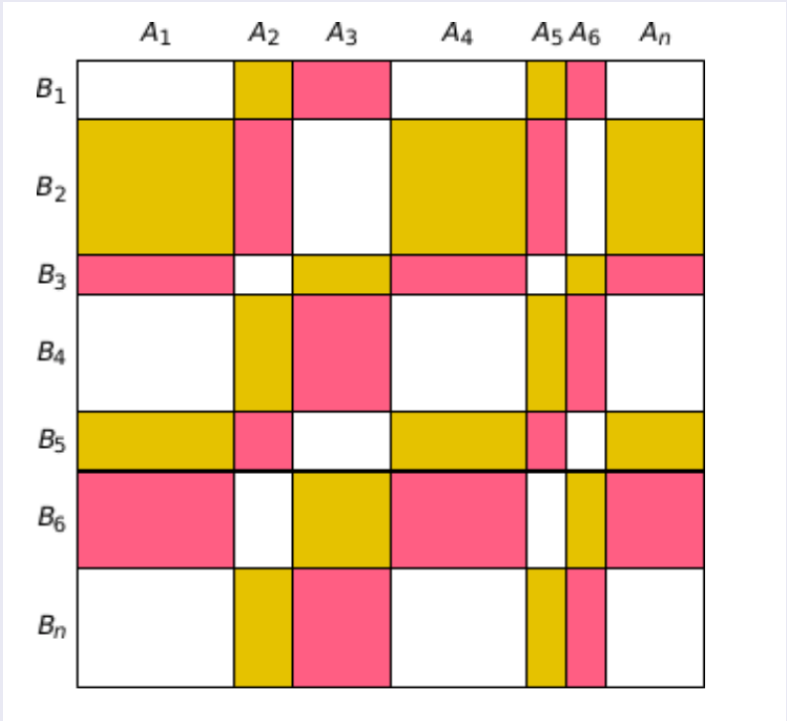
Balloon capacity goes from 1 to n, you are given the amount of helium in each of the n helium canisters. Your goal is to assign gas canisters to balloons in a way such that the least full balloon still contains the maximum possible fraction of helium inside. If a ballon is filled beyond its capacity, it will explode.

Solution

Greedy solution: Sort canisters by the amount of gas. For each $i \in \{1..n\}$, store the minimum value of $gas[i]/i$, print `impossible` if you find such a value that is greater than 1.

B - Frosting on the cake

Problem statement



How big is the area covered by each color?

NOTE: the dimensions of the grid are bounded by 10^5, so the obvious $\mathcal{O}(nm)$ solution will get you a TLE.

B - Frosting on the cake

Solution

In fact, it is enough if you iterate over one dimension at a time. First compute sums S_k of A_i, where $i \ mod \ 3 = k$, for $k = 0, 1, 2$.
Then iterate over the B_i's multiplying the S_k's and summing to the correspondent counters of the color.

NOTE 1: When implementing the solution pay attention to the fact that colors are numbered from 0 to 2, whereas the grid is indexed from 1 to n and from 1 to m.
NOTE 2: Save yourself a lot of C++ debugging using `long (long int)` numbers instead of regular 32-bit integers.

C - Cakey McCakeFace

Problem statement

You want to find the cooking time of Cakey's cakes, so you place sensors at the entry and exit points of his oven. Unluckily, sometimes these sensors didn't trigger when a cake passed under them, other times they marked a timestamp without the need to.
You want to find the cooking time that maximizes the association between plausible entry and exit timestamps.

C - Cakey McCakeFace

Problem statement

You want to find the cooking time of Cakey's cakes, so you place sensors at the entry and exit points of his oven. Unluckily, sometimes these sensors didn't trigger when a cake passed under them, other times they marked a timestamp without the need to.
You want to find the cooking time that maximizes the association between plausible entry and exit timestamps.

Solution

Compute all positive differences between input and output timestamps and use them as keys in a map. Use the value of a key as a counter to keep how many times you have encountered that cooking time. Print the smallest key with the maximum counter value.

D - Ingredients

Problem statement

You are given a dependency graph on pizza recipes. Edges are marked by the cost to add an ingredient to another plate, nodes are labelled by the "prestige" of a dish. What is the maximum prestige you can obtain by staying within a given budget?

D - Ingredients

Problem statement

You are given a dependency graph on pizza recipes. Edges are marked by the cost to add an ingredient to another plate, nodes are labelled by the "prestige" of a dish. What is the maximum prestige you can obtain by staying within a given budget?

Solution

Graphs + DP: Compute a topological sort (to find dishes of cost 0), then run a BFS to find the cost and prestige of each dish. Eventually apply 2-dimensional dynamic programming to the computed costs/gains.

Table of Contents

Table of Contents

Analogies

Imagine a physical **network** of any kind: hydraulic pipes, computer networks, city maps, electric circuits...

Each edge in this real-world graph (pipe, link, road, wire) has its own **capacity**, i.e. a maximum amount of stuff (water, data, cars, electrons) that can travel along the connection in one time unit. What we want to compute in a **maximum flow problem** is the amount of "stuff" we can have flowing in such networks between two special nodes called **source** and **sink**.

Flow network

Flow network

A graph where edges are labelled with positive capacities:

$$c : V \times V \to \mathbb{R}^+$$

such that if $(u, v) \notin E \Rightarrow c(u, v) = 0$.
Two special nodes are identified, the source s and the sink t.

Flow

Flow

Formally it is a function:

$$f : V \times V \to \mathbb{R}$$

that associates flow values to each edge.

It must have the following properties:

- **Capacity bound:** $f(u, v) \leq c(u, v)$
- **Antisimmetry:** $f(u, v) = -f(v, u)$
- **Conservation of flow:** At every node (excluding source and sink) the sum of entering and exiting flow must be 0.

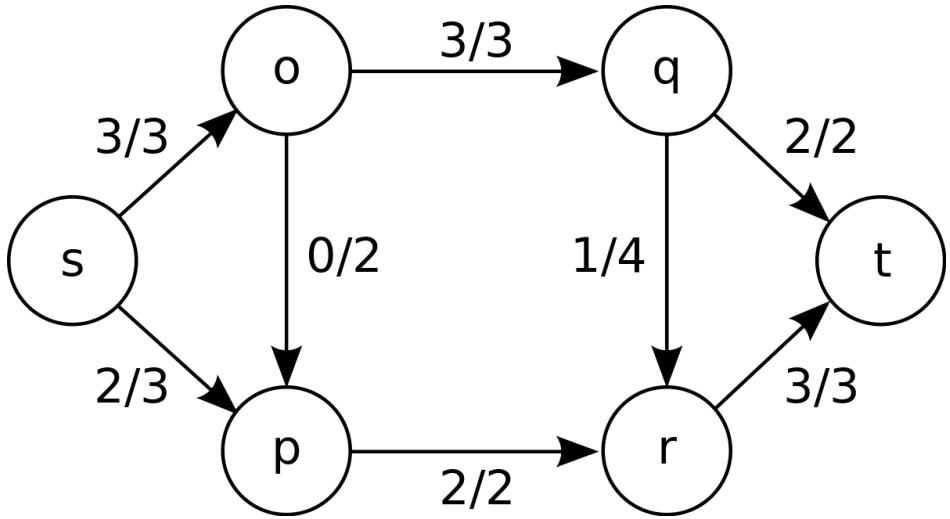
Flow value

Flow value

Once some flow has been injected into the network, we compute its value as:

$$|f| = \sum_{u \in V} f(s, u)$$

This is the flow exiting from the source s.



Source: Wikipedia

Residual network

Once some flow f has been injected into the network, we have a residual network, composed of the nodes of the original network and the edges labelled with the residual capacities $c_f(u, v) = c(u, v) - f(u, v)$

Residual network

The residual network is a structure (V, E_f, c_f, s, t), where

$$(u, v) \in E_f \Leftrightarrow c_f(u, v) > 0$$

NOTE: The original graph is a residual network where f is the empty flow.

Augmenting path

Augmenting path

An augmenting path is a positive-flow path from the source to the sink in the current residual network.

Bottlenecks

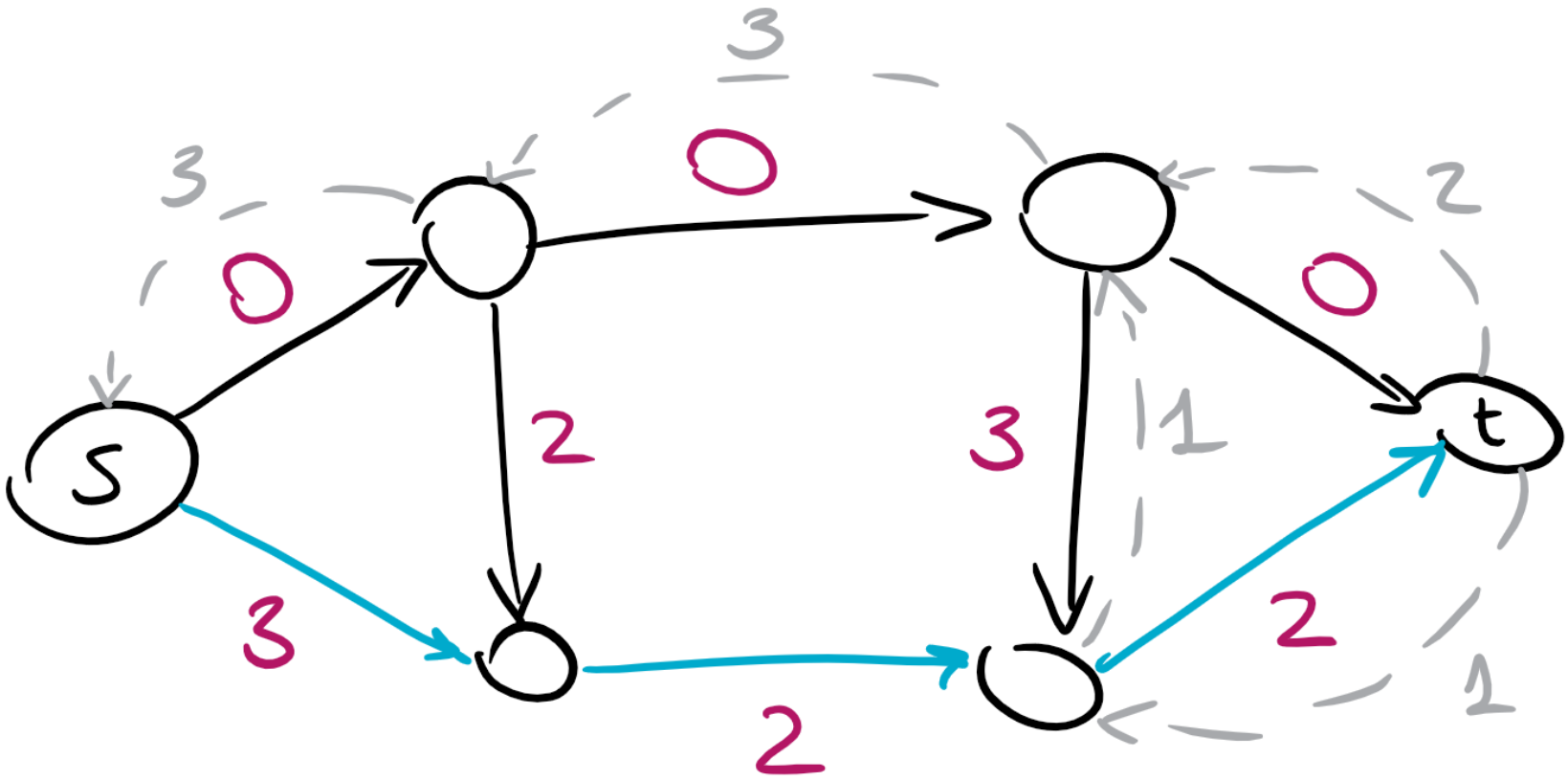
The flow along an augmenting path $\mathcal{P}$ can be at most:

$$min_{(u,v)\in P}\{c_f(u,v)\}$$

We call this edge a **bottleneck**.

Augmenting path

If we direct flow along the blue path in this residual network, its value must be at most 2, in order not to violate the capacity bound.



Why negative flow?

We need the possibility to have flow into edges of capacity 0 in order to "redirect" some flow along edges that were incorrectly added to the network by some augmenting paths.

$$c(u, v) = 0 \wedge f(u, v) < 0 \Rightarrow c_f(u, v) > 0$$

Table of Contents

Ford & Fulkerson

```
int[][] maxFlow(Graph G, Node s, Node t, int[][] c)
int[][] f = new int[][]                          % Flusso parziale
int[][] g = new int[][]                          % Flusso da cammino aumentante
int[][] r = new int[][]                          % Rete residua

foreach u, v ∈ G.V() do
    f[u][v] = 0                                  % Inizializza un flusso nullo
    r[u][v] = c[u][v]                            % Copia c in r

repeat
    g = flusso associato ad un cammino aumentante in r, oppure f_0
    foreach u, v ∈ G.V() do
        f[u][v] = f[u][v] + g[u][v]              % f = f + g
        r[u][v] = c[u][v] − f[u][v]              % Calcola c_f
until g = f_0
return f
```

Ford & Fulkerson

The visit can be in any order you like, i.e. both BFS and DFS work.

Complexity: $\mathcal{O}((V + E)|f^*|)$ (value of the maximum flow, which can be huge and/or hard to predict)

Edmonds & Karp

The smart way to search

Instead of searching for an augmenting path in any possible way (DFS or BFS) we can search for it using a BFS, thus diminishing the maximum number of such paths.

How does this work? Every search for a path costs $\mathcal{O}(V + E)$ (BFS), but there are at most $\mathcal{O}(VE)$ of suitable shortest-paths.
Why does this hold? Proof is left to the reader as exercise.

Complexity: $\mathcal{O}(VE^2)$

Table of Contents

Min-cut

Min-cut

Consider a graph $G = (V, E)$ and a function $c : E \to \mathbb{R}^+$. Partition the set V into two sets S and T, such that $s \in S$, $t \in T$ and:

$$\sum_{u \in S, v \in T} c(u, v)$$

is minimal.

Min-cut

Min-cut

Consider a graph $G = (V, E)$ and a function $c : E \to \mathbb{R}^+$. Partition the set V into two sets S and T, such that $s \in S$, $t \in T$ and:

$$\sum_{u \in S, v \in T} c(u, v)$$

is minimal.

The value of the min-cut is exactly the value of the maximum flow $|f^*|$ (Max-flow min-cut theorem).

Informally, the edges that go from S to T are the bottlenecks of the max-flow problem, i.e. those edges with residual capacity 0 after running a Edmonds-Karp.

Maximum cardinality bipartite matching

Problem definition

Consider a bipartite graph $G = (U \cup V, E)$. Match each element in U with exactly one node in V with as many edges as possible (i.e. make as many matches as possible).

How?

- Edges in E have capacity 1.
- Create a source connected to all elements in U with capacity 1
- Connect all elements in V with the sink (capacity 1).

Graph Modelling

Multi-source & multi-sink

Run Edmonds-Karp on a graph where two fresh nodes (a *super-source* and a *super-sink*) are connected by infinite-capacity edges to all sources and all sinks respectively.

Vertex capacities

Each vertex u is split into two components (u_{in}, u_{out}) connected by an edge such that:

$$c(u_{in}, u_{out}) = c(u)$$

Dinic's Algorithm (1970)

Based on the concepts of **blocking flow** and **layered network**.

Complexity: $\mathcal{O}(V^2E)$

See this tutorial for the description and proofs of correctness and complexity.