Implementation of algorithms and data structures

6. seminar

Jirka Fink

https://ktiml.mff.cuni.cz/~fink/

Department of Theoretical Computer Science and Mathematical Logic
Faculty of Mathematics and Physics
Charles University in Prague

Summer semestr 2023/24

Last change 13. listopadu 2023

Maximum flow in a network

Network (graph)

- (V, E) is a directed graph on n vertices and m edges
- $c : E \to \mathbb{R}_0^+$ is capacity of edges
- $s, t \in V$ are source and sink vertices

Flow

Flow is a function $f : E \to \mathbb{R}_0^+$ satisfying:

- Capacity constraint: $0 \leq f(e) \leq c(e)$ for every edge e
- Kirchhoff law: $\sum_{u:uv \in E} f(uv) = \sum_{u:vu \in E} f(vu)$ for every vertex v except s, t

Overflow

Overflow of a vertex v is $f^{\triangle}(v) = \sum_{u:uv \in E} f(uv) - \sum_{u:vu \in E} f(vu)$

Terminology

- Reserve (residual) of an edge uv is $r(uv) = c(uv) - f(uv) + f(vu)$
- Edge uv is saturated if $r(uv) = 0$

Wave

Wave is a function $f : E \rightarrow \mathbb{R}_0^+$ satisfying:

- Capacity constraint: $0 \leq f(e) \leq c(e)$ for every edge e
- Non-negative overflows: $f^\triangle(v) \geq 0$ for every vertex v except the source

Operation: Overflow transfer

- Transferring overflow on an edge uv means increasing $f(uv)$ by $\min\left\{f^\triangle(u), r(uv)\right\}$.
- Overflow transfer is called saturated if the reserve $r(uv)$ is reduced to zero

Height

- Height is a function $h : V \rightarrow \mathbb{Z}_0^+$
- Overflow can transferred only from a higher vertex to a lower one (downhill)
- Invariant: $h(s) = n$ a $h(t) = 0$
- Height of other vertices is initialized by 0 and algorithm can only increase it (by 1)
- If for a vertex v with $f^\triangle(v) > 0$ no overflow can be transferred, increase height $h(v)$ by one
- Improved version: From all vertices with positive overflow, choose the highest one.

Analysis of Goldberg's algorithm

$$h(v) = \begin{cases} n & \text{for } v = s \\ 0 & \text{otherwise} \end{cases}$$

$$f(uv) = \begin{cases} c(uv) & \text{for } u = s \\ 0 & \text{otherwise} \end{cases}$$

while *exists a vertex* $u \neq s, t$ *satisfying* $f^{\triangle}(u) > 0$ **do**
    if *exists an edge* uv *satisfying* $r(uv) > 0$ *and* $h(u) > h(v)$ **then**
        transfer overflow on edge uv
    else
        increase height $h(u)$ by 1

How many times each operation is called

- Finding a highest vertex with positive overflow: $O(n^2\sqrt{m})$-times
- Find non-saturated edge going downhill: $O(n^2\sqrt{m})$-times
- Saturated transfer: $O(nm)$-times
- Non-saturated transfer: $O(n^2\sqrt{m})$-times
- Increasing height: $O(n^2)$-times

Data representation

How many times each operation is called

- Finding a highest vertex with positive overflow: $O(n^2\sqrt{m})$-times
- Find non-saturated edge going downhill: $O(n^2\sqrt{m})$-times
- Saturated transfer: $O(nm)$-times
- Non-saturated transfer: $O(n^2\sqrt{m})$-times
- Increasing height: $O(n^2)$-times

Is it possible to achieve total complexity $O(n^2\sqrt{m})$?

What we need to store?

- Network which efficiently allow us
  - Obtain the current height $h(u)$ and overflow $f^\triangle(u)$
  - Obtain the current flow $f(uv)$
  - Calculate the reserve $r(uv) = c(uv) - f(uv) + f(vu)$
- The list $L(u)$ of non-saturated downhill edges from every vertex u
- A list P of vertices with positive overflow which can find a highest one

Graph representation

1st option: Pointers to opposite edges

- Structure for an edge
  - Destination vertex
  - Capacity
  - Flow
  - Pointer to the opposite edge
- Every vertex has a list of edges
- Disadvantage: there are two instances of the edge structure for every edge

2nd option: Shared edge structure for both directions

- Structure for an edge
  - Both end-vertices
  - Capacity for both directions (if they can be different)
  - Flow (negative value means flow in the opposite direction)
- Disadvantage: Auxiliary functions for handling symmetries are needed

3st option: Hash table

- Every vertex has a list of incident edges
- Hash table: (vertex,vertex) $\rightarrow$ edge data (i.e. capacity, flow)
- Disadvantage: We no longer have worst-case complexity but expected

List of non-saturated downhill edges $L(u)$

Representation

Every vertex has a list of incident vertices/edges in $L(u)$

Trivial operations in $O(1)$-time

- Test emptiness of $L(u)$
- Find an arbitrary element in $L(u)$
- Erase edge from $L(u)$ after saturated transfer

Update the list $L(u)$ after increasing height $h(u)$

- All edges incident to u can be processed in $O(\mathrm{deg}(u))$
- Height of every vertex is increased at most $2n$-times
- Complexity of all these updates:
  $\sum_{u \in V} 2nO(\mathrm{deg}(u)) = 2n \sum_{u \in V} O(\mathrm{deg}(u)) = O(nm)$

Removing the opossite edge vu from $L(v)$ when $h(u)$ is increased

Problem: find the position of vu in the list $L(v)$

- Intrusive list can find and delete in $O(1)$-time
- Lazy solution: Delete vu from $L(v)$ when the vertex v is processed

List P of vertices with positive overflow

What we need?

- Find an arbitrary vertex of P with the largest height
- Remove the vertex from P after transferring whole overflow
- Increase height of a vertex of P by one
- Insert the vertex v to P after transfer on an edge uv
  - Note that $h(v) = h(u) - 1$ in this case

Using a heap increases the complexity by $O(log(n))$-factor

Approach

- Split vertices of P into groups by their heights
- Store every group in a special list
- Access groups using a main list/array indexed by the height
- How to represent the main list?

List P of vertices with positive overflow

Approach

- Split vertices of P into groups by their heights
- Store every group in a special list
- Access groups using a main list/array indexed by the height
- How to represent the main list?

1st option: The sorted list of all non-empty groups

The highest two groups can be reached in $O(1)$-time

2nd option: Array indexed by the height with index to the highest non-empty group

After removing the last vertex from the highest group, the index has to updated which cannot be done in $O(1)$-time

- Index is always increased by one
- Total number of increment is at most $2n^2$
- If the index is decrement by ones, total number of decrements is at most $2n^2$

API: Class encapsulating all data with public functions

Public functions creating instance and obtaining results

- Add a vertex
- Add an edge with capacity
- Run the algorithm
- Get the size of a maximum flow
- Get flow on every edge

How to indentify a vertex?

- Allow only numbers from 1 to n
- All arbitrary key and internally use a hash table

How to identify an edge

- Edges can be indexed from 1 to m (impractical)
- Only iterate edges incident with a given vertex
- Internally use a hash table (vertex,vertex) $\rightarrow$ edge

API: Use external library for graph representation

Example of libraries

- Networkx in Python
- Boost in C++

API is only one function

- Argument: A graph with capacity as an edge property
- Return value: The size of a maximum flow
- The function sets a maximum flow as an edge property

Tests

Testing correctness of a solution

- Capacity constraint
- Kirchhoff law
- Saturated cut

Testing graphs

How to choose a graph?

- Small graphs, examples from literature
- Complete graphs, path, cycles, ...
- Random graphs
- Adversary graphs
  - Disconnected graphs, isolated vertices and edges
  - Graphs having dead branches accessible from the source
  - Find graphs on which the algorithm is slowest

How to choose capacities

- Unit capacity
- Regular, e.g. from 1 to m
- Random, e.g. uniform or normal distribution

Testing data consistency during the algorithm

- Correct structure of a graph (depends on representation)
- Check overflows on all vertices
- Check that f is a wave
  - Capacity constraint: $0 \leq f(e) \leq c(e)$ for every edge e
  - Non-negative overflows: $f^{\triangle}(v) \geq 0$ for every vertex v except the source
- Every vertex v satisfies $0 \leq h(v) \leq 2n$
  - except $h(z) = n$ a $h(s) = 0$
- Lists P and $L(u)$ contains only the expected vertices
- Every edge uv with $r(uv) > 0$ satisfies $h(u) - h(v) \leq 1$
- There exists a non-saturated from every vertex with positive overflow to the source
- Calculate the number of operations and potentials

Implement the algorithm step by step

General approach

- Design data representation and API and implement them
- Implement tests
- Implement algorithm
- Test and debug

Split implementation into peaces

- 1st step: Correct but slow version
  - Skip lists P anf $L(U)$
  - Functions using P and $L(U)$ are implemented trivially
  - Debug graph representation, main parts of the algorithm and all tests
- 2nd step: Implement P
- 3rd step: Implement $L(u)$

Discussion

- Can the first step be simplified?
- Can the implementation be split into more testable steps?

Start writing tests as soon as possible

Reason: Testing the tests

- Tests often contains bugs reporting non-existing bugs as well as missing bugs
- Running tests on incompletely (improperly) implemented functions verifies correctness of (some) tests
- No needs to write even more code, just run tests when implementing and check that tests fails as expected for the current code

Start writing tests as soon as possible

Examples

- Algorithm initialize heights of all vertices to 0 except $h(s) = n$
  $\Rightarrow$ Run tests before initializing heights
- Every edge uv with $r(uv) > 0$ should satisfy $h(u) - h(v) \leq 1$
  $\Rightarrow$ Initialize heights but not flows from the source
- Vertices stores their overflows
  $\Rightarrow$ Initialize flows but not overflows
- Vertices with overflows are stored in the list P
  $\Rightarrow$ Initialize flows and overflows but not P
  $\Rightarrow$ After reducing the overflow to zero, the vertex should be removed from P
- Similarly for lists of non-saturated downhill edges
- Implement increasing heights before transferring overflows
  $\Rightarrow$ The program should loop forever but tests checking the height upper bound should fail

Exercises

Theoretical questions

- What happens if the network is not connected?
- What happens if the network is connected but it is not strongly connected?
- Let A be the set of all vertices having height at least n when the algorithm terminated. Does $E(A)$ forms a saturated cut between source and sink?
- Is the source the only vertex of height n?
- What may happen if we set the height of source to be $n-1$ (or $n-2$)?
- For which graphs the algorithm requires the largest number of iterations (for fixed n or m)?

Implementation questions

- Develop unit and fuzz tests
- Based on our analysis, develop as many data consistency tests as possible
- Find data representation so that whole algorithm has complexity $O(n^2 m)$
- How to find a highest vertex with positive overflow to improve the complexity to $O(n^2\sqrt{m})$?

Tasks for next week

The first assignment: Goldberg's algorithm

- Study and understand the algorithm including analysis and complexity
- Write data representation such that all operations has expected complexity
- Write tests
- Write API