南京大学
NANJING UNIVERSITY

Dinitz' Algorithm: The Original Version and Even's Version

Author： Yefim Dinitz

Speaker： Sun Wenbo, 201830210

2022/4/15

目录

01 Introduction

02 The Original Dinitz' Algorithm

03 The Version of Shimon Even and Alon Ita

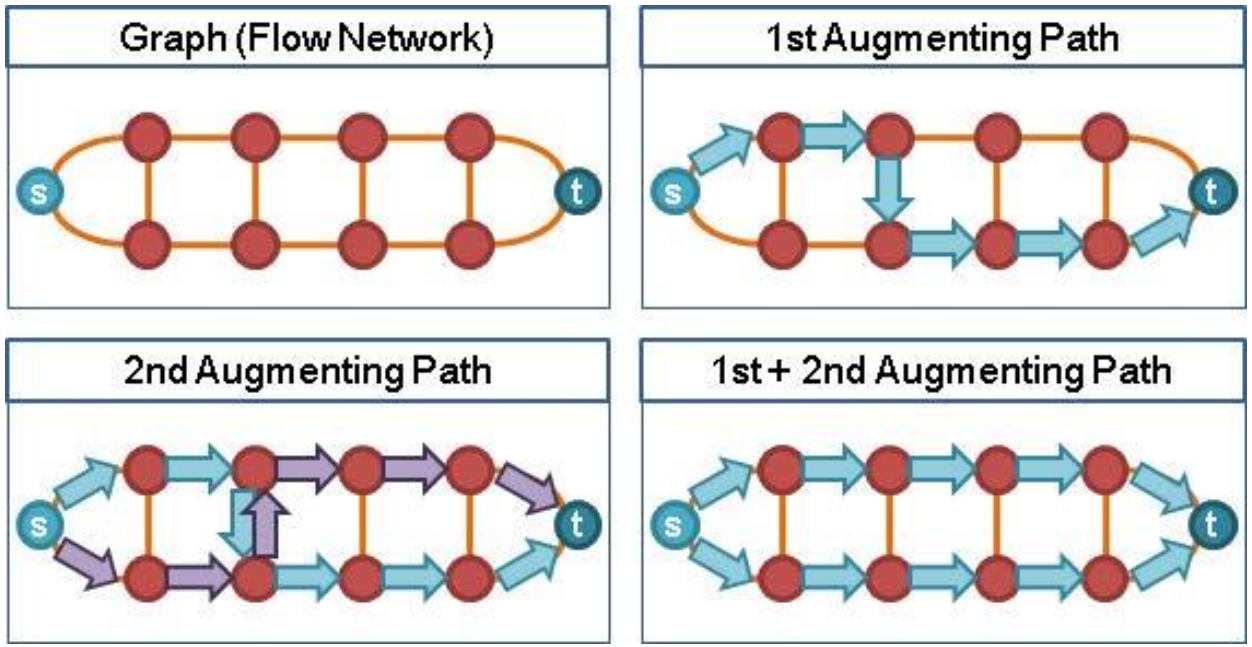
04 Implementation of DA by Cherkassky

05 Further Progress of Max-Flow Finding

01 Introduction

Introduction

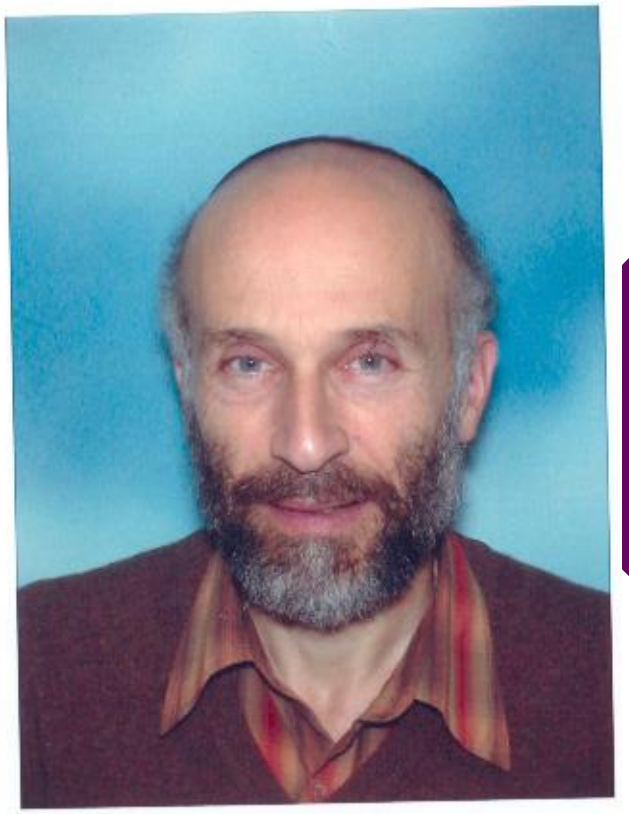
Dinic algorithm (also known as Dinitz algorithm) is an algorithm to calculate the strong polynomial complexity of the maximum flow in the network flow. It was proposed by Yefim (Chaim) a. Dinitz, a computer scientist in Israel (former Soviet Union), in 1970.



Introduction

Yefim Dinitz invented the algorithm in the pre class activity of the algorithm class of adel'son vel'sky (one of the inventors of AVL tree). At that time, he did not know the basic facts about the Ford Fulkerson algorithm.

Dinitz announced his algorithm to others in January 1969 and published it in Doklady Akademii Nauk SSSR magazine in 1970. In 1974, Shimon even and Alon Itai were interested in Dinitz's algorithm and Alexander karzanov's idea of blocking flow at the Israeli Institute of technology in Haifa.



02 The Original Dinitz' Algorithm

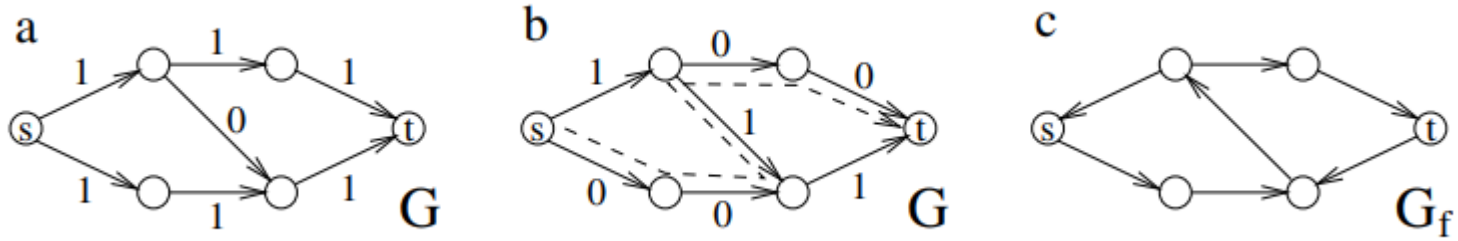
The Original Dinitz' Algorithm - max-flow problem

First, let us recall the max-flow problem definition. A capacitated directed graph G = (V,E, c), where c is a non-negative capacity function on edges, with vertices s distinguished as the source and t as the sink is given. A flow is defined as a function on the (directed) edges, f, satisfying the following two laws:

- $Capacity\ constraint$: $\forall e \in E : 0 \leq f(e) \leq c(e)$, and
- $Flow\ conservation$: $\forall v \in V \setminus \{s, t\} : \sum_{(v,u)\in E} f(v, u) - \sum_{(u,v)\in E} f(u, v) = 0.$

The quantity $\sum_{(v,u)\in E} f(v, u) - \sum_{(u,v)\in E} f(u, v)$ is called the net flow from v. The value of a flow f is defined as the net flow from s (which is equal to the negated net flow from t). The task is to find a flow of the maximal value, called **"maximum flow"**. Initially, some feasible flow function, f0, is given, which is by default the zero function.

The Original Dinitz' Algorithm – FF algorithm

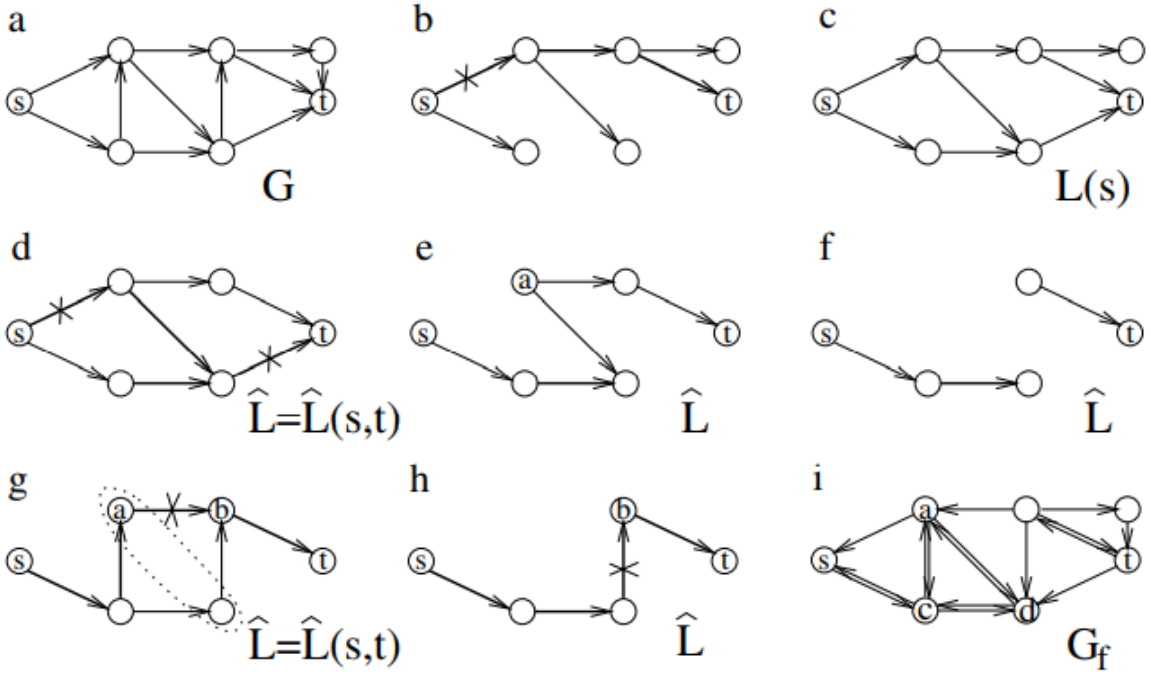


The classic **Ford&Fulkerson algorithm** (also known as FF) finds a maximum flow using the following natural idea: **augmenting the current flow iteratively, by pushing an additional amount of flow on a single source-sink path at each iteration, as long as possible.**
Let f denote the current flow. A path from s to t is sought, such that none of its edges is saturated by f; such a path is called "augmenting". When an augmenting path, P, is found, its current capacity $cf(P) = min_{e \in P} cf(e) > 0$ is computed. Then, the flow on every edge of P is increased by cf (P). As a result of this iteration, the flow value grows by *cf (P)* and at least one edge of P becomes saturated. For illustration see Figure above.

The Original Dinitz' Algorithm - Dinic algorithm

The idea of **dinic algorithm** is to expand in **Layered Network** in stages. It is different from the shortest augmented path algorithm in that: after BFS augmentation is performed once in each stage of the shortest augmented path, BFS should be restarted to find another augmented path from the source point vs; In dinic algorithm, multiple augmentation can be realized only by one DFS process, which is the cleverness of dinic algorithm. The specific steps of dinic algorithm are as follows:

(1) Initialize network capacity and network flow.
(2) Construct residual network and hierarchical network. If the sink is no longer in the hierarchical network, the algorithm ends.
(3) In the hierarchical network, **a DFS process** is used for augmentation. DFS is completed and the augmentation of this stage is also completed.
(4) Go to step (2).



The Original Dinitz' Algorithm - General Description

PathFinding: Starting from t, repeatedly choose one of the incoming edges and pass to its tail. Necessarily, after steps the layer 0, that is s, is reached. The path consisting of the chosen edges (in the reverse order), is a shortest path from s to t.

RightPass For each edge e ∈ Qr, if its right end-vertex has no incoming edges, then it is deleted from L^, together with all its outgoing edges, while inserting those edges into Qr.

LeftPass For each edge e ∈ Ql , if its left end-vertex has no outgoing edges, then it is deleted from L^, together with all its incoming edges, while inserting those edges into Ql.

The Original Dinitz Algorithm

Input:
a flow network $G = (V, E, c, s, t)$,
a feasible flow f, in G (equal to zero, by default).

```
/* Phase Loop: */
dowhile
begin
    Build L̂(s,t) in G_f, using the extended BFS;
    if L̂(s,t) = ∅ then return f
    else L̂ ← L̂(s,t);
    /* Iteration Loop: */
    while L̂ is not empty do
    /* Iteration Invariant: L̂ is the union of all shortest augmenting paths */
    begin
        P ← PathFinding(L̂);
        Sat ← FlowChange(P);
        /* Cleaning(L̂): */
        begin
            Removal of edges in Sat;
            Q^r, Q^l ← Sat;
            RightPass(Q^r);
            LeftPass(Q^l);
        end;
    end;
end;
```

The Original Dinitz' Algorithm - Analysis of dinic algorithm

The Polynomial Time Bound of DA

An easy consequence of the foregoing property is that there are at most |V | − 1 phases in DA, since the distance from s to t cannot exceed |V | − 1.

It is easy to see that the number of iterations during a single phase is at most |E|, since at least one out of at most |E| edges of L is removed from it at every iteration. Therefore, the total running time at each phase, which consists of the accelerated iteration times and the layered network building and maintenance time, is O(|E|·|V | + |E|) = O(|V ||E|). Thus, DA is finite and its total running time is O($|\boldsymbol{V}|^2$|E|). We arrive at our main result:

The Dinitz algorithm builds a maximum flow in time O($|\boldsymbol{V}|^2$|E|).

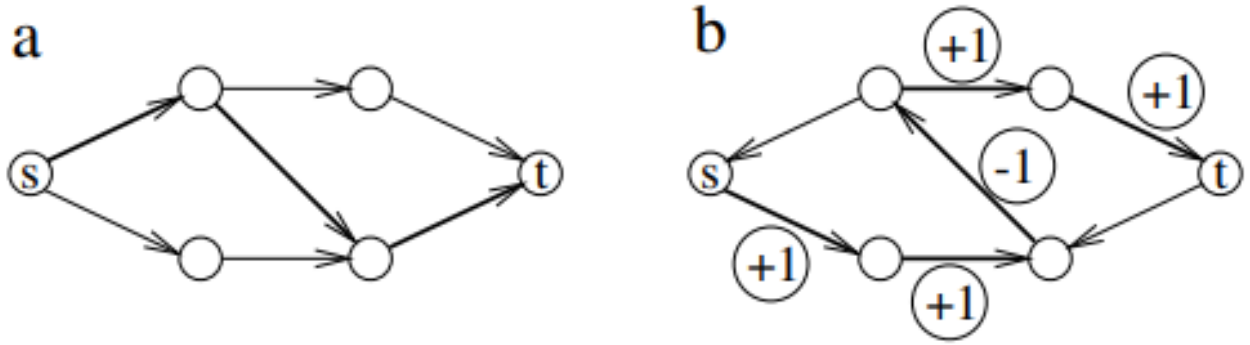
03 The Version of Shimon Even and Alon Ita

The Version of Shimon Even and Alon Ita

Even and Itai go much farther while changing DA. They admit dead-ends in L in both directions; thus, they cancel Cleaning and are urged to give up using **PathFinding**. They suggest another way to find an augmenting path: a search of the DFS type, combined with encountered dead-ends removal. Each iteration of their version of DA is as follows.

Their DFS begins to build a path from s, incrementing it edge by edge, from one layer to the next. Such path building stops either by arrival at t or at another dead-end. In the last case, DFS **backtracks to the tail of the edge leading to that dead-end**, while **removing that edge from L as useless** (i.e. not contained in any path from s to t in L). After that, DFS continues as above, incrementing the current path when possible or backtracking and removing the last edge from L, otherwise. This process ends either when DFS is at s and has no edge to leave it, indicating that the phase is finished, or when it arrives at t. In the latter case, an augmenting path is built. Then, a flow change is executed along that path, as at FF, while removing from L all its edges which have become saturated; recall that there is at least one such edge.

The Version of Shimon Even and Alon Ita



The running time of a phase, after executing the extended BFS to initialize L, is counted by Even and Itai as divided into intervals between arrivals at dead-ends. First, each such event causes the removal of an edge from L, so there may be at most |E| such events. Secondly, there may be at most forward steps of DFS between neighboring events, which costs O(l). Except for those steps, there may be either a single backtracking and edge removal, which costs O(1), or a flow change, together with saturated edge removals, which costs O(l). In any case, every interval between consequent events costs O(l), which totals O(l·|E|) = O(|V||E|) per phase, as desired.

04 Implementation of DA by Cherkassky

Implementation of DA by Cherkassky

From the 1970s, Boris Cherkassky worked on the quality implementation of various flow algorithms and on the experimental evaluation of their performance. His recommendations for the best implementation of DA are as follows:

- No layered network—neither L(s) nor Lˆ(s, t)—should be built. It is sufficient to compute just the layer number dist(v, t) for every vertex up to the distance = dist(s, t). This may be done by a single run of the usual BFS from t on the unsaturated edges, in the inverse edge direction.
- The entire phase is conducted by a single DFS from s. Any saturated edge or edge not going from a vertex of rank i to a vertex of rank i + 1 is just skipped (instead of using the list of edges in the layered network).
- ……

Implementation of DA by Cherkassky

```
/* Input: */
    a flow network G = (V, E, c, s, t),
    a feasible flow f, in G (equal to zero, by default).
Initialization:
    compute ∀e ∈ E : c_f(e) = c(e) − f(e);
/* Phase Loop: */
    dowhile
    begin
        compute ∀v ∈ V : rank(v) = dist(v, t), by BFS from t on edges
          with c_f > 0, in the inverse edge direction;
        if rank(s) = ∞ then begin f ← c − c_f; return f; end;
        while DFS from s do
        begin
            /* P denotes the current path and x the current vertex of DFS */
            any edge (x, y) s.t. c_f(x, y) = 0 or rank(y) ≠ rank(x) − 1 is skipped;
            if x = t then
            begin
                ε ← min{c_f(e) : e ∈ P};
                for edges (v, u) of P, from t downto s do
                begin
                    c_f(v, u) ← c_f(v, u) − ε; c_f(u, v) ← c_f(u, v) + ε;
                    if c_f(u, v) = 0 then x ← u;
                end;
            /* continue DFS from x */
            end;
        end;
    end;
```

05 Further Progress of Max-Flow Finding

Further Progress of Max-Flow Finding

Edmonds' and Karp's Version of FF

Edmonds and Karp proved the finiteness and polynomial time of FF, if only the shortest augmenting paths are used in it. The achieved time bound is O(|V||E|^2) = O(|V|^5), which is higher than O(|V|^2|E|) = O(|V|^4) of DA, since iterations are made in O(|E|) time each, as usual in FF.

Karzanov's Algorithm

Karzanov was the first to leave the augmenting paths idea, in the research on max-flow finding. He suggested the "preflow-push" approach, as follows. He observed that at each phase of DA, it is sufficient to somehow find a flow in a layered network Lˆ(s, t), such that any path from s to t contains an edge saturated by it; he calls such a flow "blocking". His algorithm (henceforth, KA) finds a blocking flow in O(|V|^2) time, thus arriving at an O(|V|^3) max-flow algorithm.

Further Progress of Max-Flow Finding

Improvements upon DA and KA

Much effort was made, based on DA and KA, to lower the running time bound of max-flow finding; the aim was to acquire an O(|V||E|) max-flow algorithm. Following is a brief review, cut before the push-relabel algorithm.

Cherkassky first simplified KA and later suggested a “hybrid” of DA and KA running in time $O(|V|^2\sqrt{|E|})$. Galil accelerated his algorithm to run in time O(|V|^5/3|E|^2/3).

Galil and Naaman suggested an acceleration of augmenting path finding in DA by means of storing parts of previously found augmenting paths.

Shiloach discovered a similar algorithm independently a bit later. Their algorithms run in time O(|E||V|log^2(|V|)), which differs from the “goal time” only in a poly-logarithmic factor.

Sleator and Tarjan improved the algorithm from to run in time O(|E||V|log(|V|)) by using trees instead of just paths.

These low running times were achieved by using smart, but heavy, data structures.

Acknowledgement

Thanks for watching!