Design and Analysis of Algorithms (IX)

More Extensions on Ford-Fulkerson Algorithm

Guoqiang Li
School of Software

SHANGHAI JIAO TONG UNIVERSITY

Capacity-Scaling Algorithm

Capacity-Scaling Algorithm

Overview. Choosing augmented paths with large bottleneck capacity.

Capacity-Scaling Algorithm

Shanghai Jiao Tong University

Overview. Choosing augmented paths with large bottleneck capacity.

- Maintain scaling parameter Δ.

Capacity-Scaling Algorithm

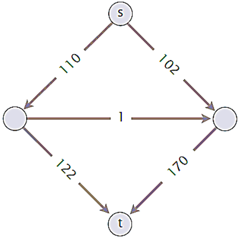
Overview. Choosing augmented paths with large bottleneck capacity.

- Maintain scaling parameter Δ.
- Let $G_f(\Delta)$ be the part of the residual network containing only those edges with capacity $\geq \Delta$.

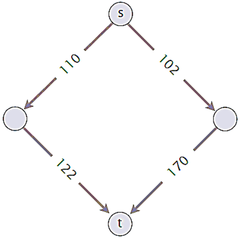
Capacity-Scaling Algorithm

Overview. Choosing augmented paths with large bottleneck capacity.

- Maintain scaling parameter Δ.
- Let $G_f(\Delta)$ be the part of the residual network containing only those edges with capacity $\geq \Delta$.
- Any augmenting path in $G_f(\Delta)$ has bottleneck capacity $\geq \Delta$.



G_f



$G_f(\Delta)$, $\Delta = 100$

Capacity-Scaling Algorithm

SHANGHAI JIAO TONG UNIVERSITY

```
CAPACITY-SCALING(G)

for each edge e ∈ E do
 |  f(e) ← 0
end
Δ ← largest power of 2 ≤ C;
while Δ ≥ 1 do
 |  G_f(Δ) ← Δ-residual network of G with respect to flow f;
 |  while there exists an s ⇝ t path P in G_f(Δ) do
 |   |  f ← AUGMENT(f, P);
 |   |  UPDATE(G_Δ(f));
 |  end
 |  Δ = Δ/2;
end
RETURN f;
```

Proof of Correctness

Assumption. All edge capacities are integers between 1 and C.

Proof of Correctness

Assumption. All edge capacities are integers between 1 and C.

Invariant. The scaling parameter Δ is a power of 2.

Proof of Correctness

Assumption. All edge capacities are integers between 1 and C.

Invariant. The scaling parameter Δ is a power of 2.

Proof. Initially a power of 2; each phase divides Δ by exactly 2.

Proof of Correctness

Assumption. All edge capacities are integers between 1 and C.

Invariant. The scaling parameter Δ is a power of 2.

Proof. Initially a power of 2; each phase divides Δ by exactly 2.

Integrality invariant. Throughout the algorithm, every edge flow $f(e)$ and residual capacity $c_f(e)$ is an integer.

Proof of Correctness

Assumption. All edge capacities are integers between 1 and C.

Invariant. The scaling parameter Δ is a power of 2.

Proof. Initially a power of 2; each phase divides Δ by exactly 2.

Integrality invariant. Throughout the algorithm, every edge flow $f(e)$ and residual capacity $c_f(e)$ is an integer.

Proof. Same as for generic Ford–Fulkerson.

Proof of Correctness

Theorem

If capacity-scaling algorithm terminates, then f is a max flow.

Proof of Correctness

Theorem

If capacity-scaling algorithm terminates, then f is a max flow.

Proof.

Proof of Correctness

Theorem

If capacity-scaling algorithm terminates, then f is a max flow.

Proof.

- By integrality invariant, when $\Delta = 1 \Rightarrow G_f(\Delta) = G_f$
- Upon termination of $\Delta = 1$ phase, there are no augmenting paths.
- Result follows augmenting path theorem.

Analysis of Running Time

SHANGHAI JIAO TONG UNIVERSITY

Lemma 1

There are $1 + \lfloor \log_2 C \rfloor$ scaling phases.

Lemma 2

There are $\leq 2|E|$ augmentations per scaling phase.

Lemma 3

Let f be the flow at the end of a Δ-scaling phase. Then

$$\mathrm{val}(f^*) \leq \mathrm{val}(f) + |E| \cdot \Delta$$

Analysis of Running Time

Theorem

The capacity-scaling algorithm takes $O(|E|^2 \cdot \log C)$ *time.*

Analysis of Running Time

Theorem

The capacity-scaling algorithm takes $O(|E|^2 \cdot \log C)$ *time.*

Proof.

Analysis of Running Time

SHANGHAI JIAO TONG UNIVERSITY

Theorem

The capacity-scaling algorithm takes $O(|E|^2 \cdot \log C)$ *time.*

Proof.

- Lemma 1+ Lemma 2 $\Rightarrow O(|E| \cdot \log C)$ augmentations.
- Finding an augmenting path takes $O(|E|)$ time.

Analysis of Running Time

SHANGHAI JIAO TONG UNIVERSITY

Lemma 1

There are $1 + \lfloor \log_2 C \rfloor$ scaling phases.

Analysis of Running Time

SHANGHAI JIAO TONG UNIVERSITY

Lemma 1

There are $1 + \lfloor \log_2 C \rfloor$ scaling phases.

Proof.

Initially $C/2 < \Delta \leq C$; Δ decreases by a factor of 2 in each iteration.

Analysis of Running Time

Lemma 2

There are $\leq 2|E|$ augmentations per scaling phase.

Analysis of Running Time

Lemma 2

There are $\leq 2|E|$ augmentations per scaling phase.

Proof.

Analysis of Running Time

Lemma 2

There are $\leq 2|E|$ augmentations per scaling phase.

Proof.

- Let f be the flow at the beginning of a Δ-scaling phase.
- Lemma 3 $\Rightarrow \mathrm{val}(f^*) \leq \mathrm{val}(f) + |E| \cdot (2\Delta)$.
- Each augmentation in a Δ-phase increases $\mathrm{val}(f)$ by at least Δ.

Analysis of Running Time

Lemma 3

Let f be the flow at the end of a Δ-scaling phase. Then

$$\mathrm{val}(f^*) \le \mathrm{val}(f) + |E| \cdot \Delta$$

Analysis of Running Time

Lemma 3

Let f be the flow at the end of a Δ-scaling phase. Then

$$\mathrm{val}(f^*) \leq \mathrm{val}(f) + |E| \cdot \Delta$$

Proof.

Analysis of Running Time

Lemma 3

Let f be the flow at the end of a Δ-scaling phase. Then

$$\text{val}(f^*) \leq \text{val}(f) + |E| \cdot \Delta$$

Proof.

- We show there exists a cut (A, B) such that $cap(A, B) \leq \text{val}(f) + |E| \cdot \Delta$.
- Choose A to be the set of nodes reachable from s in $G_f(\Delta)$.
- By definition of $A : s \in A$.
- By definition of flow $f : t \notin A$.

Analysis of Running Time

Lemma 3

Let f be the flow at the end of a Δ-scaling phase. Then

$$\mathrm{val}(f^*) \le \mathrm{val}(f) + |E| \cdot \Delta$$

Proof.

Analysis of Running Time

SHANGHAI JIAO TONG UNIVERSITY

Lemma 3

Let f be the flow at the end of a Δ-scaling phase. Then

$$\text{val}(f^*) \le \text{val}(f) + |E| \cdot \Delta$$

Proof.

$$\begin{aligned} \text{val}(f) &= \sum_{e \text{ out of } A} f(e) - \sum_{e \text{ in to } A} f(e) \\ &\ge \sum_{e \text{ out of } A} (c(e) - \Delta) - \sum_{e \text{ in to } A} \Delta \\ &\ge \sum_{e \text{ out of } A} c(e) - \sum_{e \text{ out of } A} \Delta - \sum_{e \text{ in to } A} \Delta \\ &\ge \text{cap}(A, B) - |E| \cdot \Delta \end{aligned}$$

Shortest Augmenting Paths

Edmonds–Karp's Algorithm

Q. How to choose next augmenting path in Ford–Fulkerson?

Edmonds–Karp's Algorithm

Q. How to choose next augmenting path in Ford–Fulkerson?

A. Pick one that uses the fewest edges.

Edmonds–Karp's Algorithm

Q. How to choose next augmenting path in Ford–Fulkerson?

A. Pick one that uses the fewest edges.

EDMONDS–KARP'S ALGORITHM(G)

for *each edge* $e \in E$ **do**
  $f(e) \leftarrow 0$
end
$G_f \leftarrow$ residual network of G with respect to flow f;
while *there exists an* $s \rightsquigarrow t$ *path in* G_f **do**
  $P \leftarrow$ BFS(G_f) ;
  $f \leftarrow$ AUGMENT(f, P) ;
  UPDATE(G_f) ;
end
RETURN f;

Overview of Analysis

Lemma 1

The length of a shortest augmenting path never decreases.

Overview of Analysis

Lemma 1

The length of a shortest augmenting path never decreases.

Lemma 2

After at most $|E|$ shortest-path augmentations, the length of a shortest augmenting path strictly increases.

Overview of Analysis

Lemma 1

The length of a shortest augmenting path never decreases.

Lemma 2

After at most $|E|$ shortest-path augmentations, the length of a shortest augmenting path strictly increases.

Theorem

The Edmonds–Karp's algorithm takes $O(|E|^2|V|)$ *time.*

Overview of Analysis

Lemma 1

The length of a shortest augmenting path never decreases.

Lemma 2

After at most $|E|$ shortest-path augmentations, the length of a shortest augmenting path strictly increases.

Theorem

The Edmonds–Karp's algorithm takes $O(|E|^2|V|)$ time.

Proof.

Overview of Analysis

Lemma 1

The length of a shortest augmenting path never decreases.

Lemma 2

After at most $|E|$ shortest-path augmentations, the length of a shortest augmenting path strictly increases.

Theorem

The Edmonds–Karp's algorithm takes $O(|E|^2|V|)$ time.

Proof.

- $O(|E|)$ time to find a shortest augmenting path via BFS.

Overview of Analysis

SHANGHAI JIAO TONG UNIVERSITY

Lemma 1

The length of a shortest augmenting path never decreases.

Lemma 2

After at most $|E|$ shortest-path augmentations, the length of a shortest augmenting path strictly increases.

Theorem

The Edmonds–Karp's algorithm takes $O(|E|^2|V|)$ time.

Proof.

- $O(|E|)$ time to find a shortest augmenting path via BFS.
- There are $\leq |V||E|$ augmentations.

Overview of Analysis

Lemma 1

The length of a shortest augmenting path never decreases.

Lemma 2

After at most $|E|$ shortest-path augmentations, the length of a shortest augmenting path strictly increases.

Theorem

The Edmonds–Karp's algorithm takes $O(|E|^2|V|)$ time.

Proof.

- $O(|E|)$ time to find a shortest augmenting path via BFS.
- There are $\leq |V||E|$ augmentations.
  - at most $|E|$ augmenting paths of length k

Overview of Analysis

Lemma 1

The length of a shortest augmenting path never decreases.

Lemma 2

After at most $|E|$ shortest-path augmentations, the length of a shortest augmenting path strictly increases.

Theorem

The Edmonds–Karp's algorithm takes $O(|E|^2|V|)$ time.

Proof.

- $O(|E|)$ time to find a shortest augmenting path via BFS.
- There are $\leq |V||E|$ augmentations.
  - at most $|E|$ augmenting paths of length $k \longleftarrow$ Lemma 1 + Lemma 2

Overview of Analysis

Lemma 1

The length of a shortest augmenting path never decreases.

Lemma 2

After at most $|E|$ shortest-path augmentations, the length of a shortest augmenting path strictly increases.

Theorem

The Edmonds–Karp's algorithm takes $O(|E|^2|V|)$ time.

Proof.

- $O(|E|)$ time to find a shortest augmenting path via BFS.
- There are $\leq |V||E|$ augmentations.
  - at most $|E|$ augmenting paths of length $k \longleftarrow$ Lemma 1 + Lemma 2
  - at most $|V| - 1$ different lengths.

Edmonds–Karp's Algorithm: Analysis

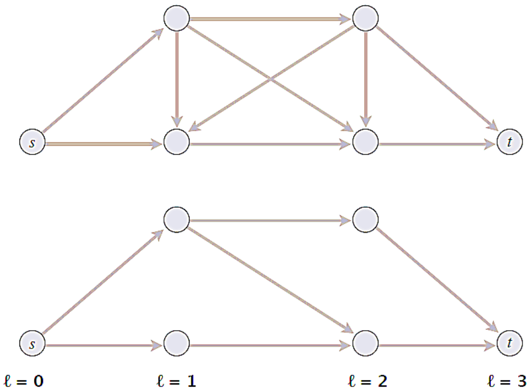
Shanghai Jiao Tong University

Definition

Given a digraph $G = (V, E)$ with source s, its level graph is defined by:

- $\ell(v)$ = number of edges in shortest $s \rightsquigarrow v$ path.
- $L_G = (V, E_G)$ is the subgraph of G that contains only those edges $(v, w) \in E$ with $\ell(w) = \ell(v) + 1$.

Edmonds–Karp's Algorithm: Analysis



Quiz 5

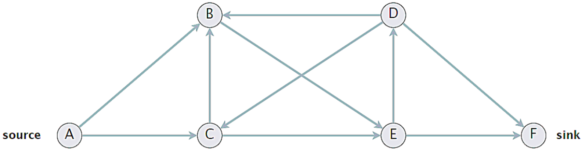
Which edges are in the level graph of the following digraph?

A. $D \to F$

B. $E \to F$

C. Both A and B.

D. Neither A nor B.



Edmonds–Karp's Algorithm: Analysis

SHANGHAI JIAO TONG UNIVERSITY

Definition

Given a digraph $G = (V, E)$ with source s, its level graph is defined by:

- $\ell(v)$ = number of edges in shortest $s \rightsquigarrow v$ path.
- $L_G = (V, E_G)$ is the subgraph of G that contains only those edges $(v, w) \in E$ with $\ell(w) = \ell(v) + 1$.

Edmonds–Karp's Algorithm: Analysis

SHANGHAI JIAO TONG UNIVERSITY

Definition

Given a digraph $G = (V, E)$ with source s, its level graph is defined by:

- $\ell(v)$ = number of edges in shortest $s \rightsquigarrow v$ path.
- $L_G = (V, E_G)$ is the subgraph of G that contains only those edges $(v, w) \in E$ with $\ell(w) = \ell(v) + 1$.

Key property. P is a shortest $s \rightsquigarrow v$ path in G iff P is an $s \rightsquigarrow v$ path in L_G.

Edmonds–Karp's Algorithm: Analysis

Lemma 1

The length of a shortest augmenting path never decreases.

Edmonds–Karp's Algorithm: Analysis

Lemma 1

The length of a shortest augmenting path never decreases.

Proof.

Edmonds–Karp's Algorithm: Analysis

Lemma 1

The length of a shortest augmenting path never decreases.

Proof.

- Let f and f' be flow before and after a shortest-path augmentation.
- Let L_G and $L_{G'}$ be level graphs of G_f and $G_{f'}$.
- Only back edges added to $G_{f'}$
  (any $s \rightsquigarrow t$ path that uses a back edge is longer than previous length)

Edmonds–Karp's Algorithm: Analysis

Lemma 2

After at most $|E|$ shortest-path augmentations, the length of a shortest augmenting path strictly increases.

Edmonds–Karp's Algorithm: Analysis

Shanghai Jiao Tong University

Lemma 2

After at most $|E|$ shortest-path augmentations, the length of a shortest augmenting path strictly increases.

Proof.

Edmonds–Karp's Algorithm: Analysis

SHANGHAI JIAO TONG UNIVERSITY

Lemma 2

After at most $|E|$ shortest-path augmentations, the length of a shortest augmenting path strictly increases.

Proof.

- At least one (bottleneck) edge is deleted from L_G per augmentation.
- No new edge added to L_G until shortest path length strictly increases.

Review of Analysis

SHANGHAI JIAO TONG UNIVERSITY

Lemma 1

The length of a shortest augmenting path never decreases.

Lemma 2

After at most $|E|$ shortest-path augmentations, the length of a shortest augmenting path strictly increases.

Theorem

The Edmonds–Karp's algorithm takes $O(|E|^2|V|)$ *time.*

Improving the Running Time

Shanghai Jiao Tong University

Note. $\Theta(|E||V|)$ augmentations necessary for some flow networks.

Improving the Running Time

Note. $\Theta(|E||V|)$ augmentations necessary for some flow networks.

- Try to decrease time per augmentation instead.
- Simple idea $\Rightarrow O(|E||V|^2)$ [Dinitz 1970]

Improving the Running Time

SHANGHAI JIAO TONG UNIVERSITY

Note. $\Theta(|E||V|)$ augmentations necessary for some flow networks.

- Try to decrease time per augmentation instead.
- Simple idea $\Rightarrow O(|E||V|^2)$ [Dinitz 1970]
- Dynamic trees $\Rightarrow O(|E||V|\log|V|)$ [Sleator–Tarjan 1983]

Dinitz' Algorithm

Dinitz' Algorithm

SHANGHAI JIAO TONG UNIVERSITY

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

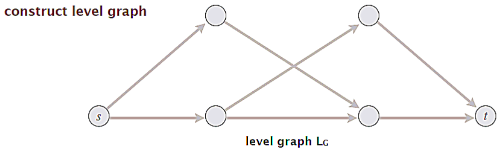
Dinitz' Algorithm

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Construct level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment flow; update L_G; and restart from s.
- If get stuck, delete node from L_G and retreat to previous node.



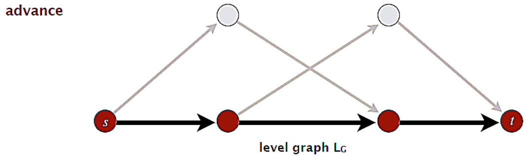
Dinitz' Algorithm

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Construct level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment flow; update L_G; and restart from s.
- If get stuck, delete node from L_G and retreat to previous node.



Dinitz' Algorithm

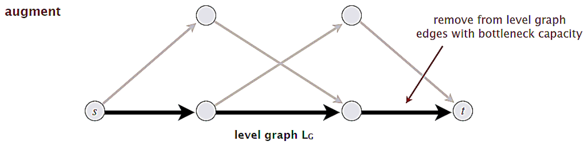
Shanghai Jiao Tong University

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Construct level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment flow; update L_G; and restart from s.
- If get stuck, delete node from L_G and retreat to previous node.



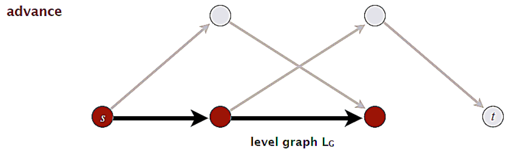
Dinitz' Algorithm

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Construct level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment flow; update L_G; and restart from s.
- If get stuck, delete node from L_G and retreat to previous node.



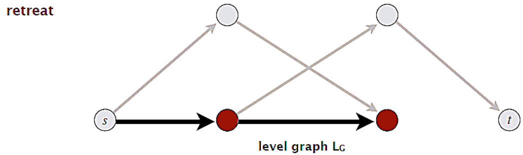
Dinitz' Algorithm

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Construct level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment flow; update L_G; and restart from s.
- If get stuck, delete node from L_G and retreat to previous node.



Dinitz' Algorithm

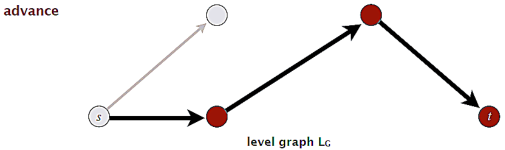
Shanghai Jiao Tong University

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Construct level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment flow; update L_G; and restart from s.
- If get stuck, delete node from L_G and retreat to previous node.



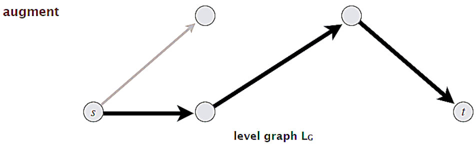
Dinitz' Algorithm

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Construct level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment flow; update L_G; and restart from s.
- If get stuck, delete node from L_G and retreat to previous node.



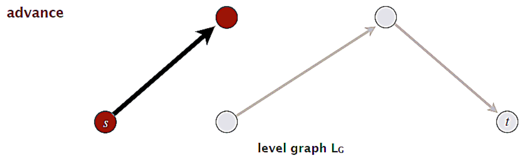
Dinitz' Algorithm

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Construct level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment flow; update L_G; and restart from s.
- If get stuck, delete node from L_G and retreat to previous node.



Dinitz' Algorithm

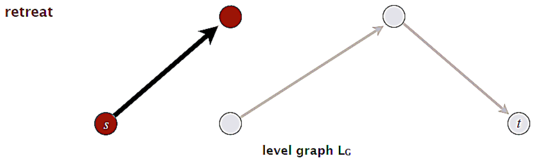
SHANGHAI JIAO TONG UNIVERSITY

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Construct level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment flow; update L_G; and restart from s.
- If get stuck, delete node from L_G and retreat to previous node.



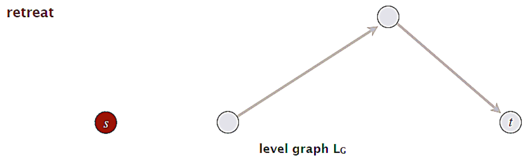
Dinitz' Algorithm

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Construct level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment flow; update L_G; and restart from s.
- If get stuck, delete node from L_G and retreat to previous node.



Dinitz' Algorithm

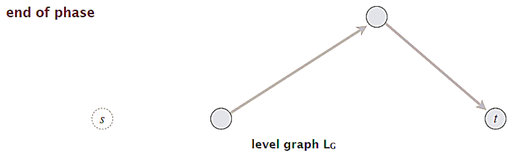
Shanghai Jiao Tong University

Two types of augmentations.

- Normal: length of shortest path does not change.
- Special: length of shortest path strictly increases.

Phase of normal augmentations.

- Construct level graph L_G.
- Start at s, advance along an edge in L_G until reach t or get stuck.
- If reach t, augment flow; update L_G; and restart from s.
- If get stuck, delete node from L_G and retreat to previous node.



Dinitz' Algorithm

```
INITIALIZE(G, f)

L_G ← level-graph of G_f
P ← ∅
GOTO ADVANCE(s) ;


RETREAT(v)

if v = s then Stop;
else
    Delete v from L_G;
    Remove last edge (u, v)
      from P;
end
GOTO ADVANCE(u) ;
```

```
ADVANCE(v)

if v = t then
    AUGMENT(P) ;
    Remove saturated edges
      from L_G;
    P ← ∅;
    GOTO ADVANCE(s) ;
end
if there exists edge (v, w) ∈ L_G
  then
    Add edge (v, w) to P;
    GOTO ADVANCE(w) ;
end
else
    GOTO RETREAT(v) ;
end
```

Quiz 6

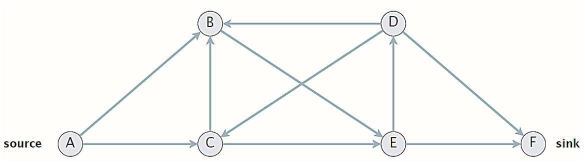
How to compute the level graph L_G efficiently?

A. Depth-first search.

B. Breadth-first search.

C. Both A and B.

D. Neither A nor B.



Dinitz' Algorithm: Analysis

Lemma

A phase can be implemented to run in $O(|E||V|)$ time.

Dinitz' Algorithm: Analysis

Lemma

A phase can be implemented to run in $O(|E||V|)$ time.

Proof.

- Initialization happens once per phase.

Dinitz' Algorithm: Analysis

Lemma

A phase can be implemented to run in $O(|E||V|)$ time.

Proof.

- Initialization happens once per phase. using BFS

Dinitz' Algorithm: Analysis

Lemma

A phase can be implemented to run in $O(|E||V|)$ time.

Proof.

- Initialization happens once per phase. using BFS
- At most $|E|$ augmentations per phase.

Dinitz' Algorithm: Analysis

Lemma

A phase can be implemented to run in $O(|E||V|)$ time.

Proof.

- Initialization happens once per phase. using BFS
- At most $|E|$ augmentations per phase. $\longleftarrow O(|E|)$ per phase

Dinitz' Algorithm: Analysis

SHANGHAI JIAO TONG UNIVERSITY

Lemma

A phase can be implemented to run in $O(|E||V|)$ time.

Proof.

- Initialization happens once per phase. using BFS
- At most $|E|$ augmentations per phase. $\longleftarrow O(|E|)$ per phase
  (because an augmentation deletes at least one edge from L_G)

Dinitz' Algorithm: Analysis

Lemma

A phase can be implemented to run in $O(|E||V|)$ time.

Proof.

- Initialization happens once per phase. using BFS
- At most $|E|$ augmentations per phase. $\longleftarrow O(|E|)$ per phase
  (because an augmentation deletes at least one edge from L_G)
- At most $|V|$ retreats per phase.

Dinitz' Algorithm: Analysis

SHANGHAI JIAO TONG UNIVERSITY

Lemma

A phase can be implemented to run in $O(|E||V|)$ time.

Proof.

- Initialization happens once per phase. using BFS
- At most $|E|$ augmentations per phase. $\longleftarrow O(|E|)$ per phase
  (because an augmentation deletes at least one edge from L_G)
- At most $|V|$ retreats per phase. $\longleftarrow O(|E| + |V|)$ per phase

Dinitz' Algorithm: Analysis

SHANGHAI JIAO TONG UNIVERSITY

Lemma

A phase can be implemented to run in $O(|E||V|)$ time.

Proof.

- Initialization happens once per phase. using BFS
- At most $|E|$ augmentations per phase. $\longleftarrow O(|E|)$ per phase
  (because an augmentation deletes at least one edge from L_G)
- At most $|V|$ retreats per phase. $\longleftarrow O(|E| + |V|)$ per phase
  (because a retreat deletes one node from L_G)

Dinitz' Algorithm: Analysis

Lemma

A phase can be implemented to run in $O(|E||V|)$ time.

Proof.

- Initialization happens once per phase. using BFS
- At most $|E|$ augmentations per phase. $\longleftarrow O(|E|)$ per phase (because an augmentation deletes at least one edge from L_G)
- At most $|V|$ retreats per phase. $\longleftarrow O(|E| + |V|)$ per phase (because a retreat deletes one node from L_G)
- At most $|E||V|$ advances per phase.

Dinitz' Algorithm: Analysis

SHANGHAI JIAO TONG UNIVERSITY

Lemma

A phase can be implemented to run in $O(|E||V|)$ time.

Proof.

- Initialization happens once per phase. using BFS
- At most $|E|$ augmentations per phase. $\longleftarrow O(|E|)$ per phase (because an augmentation deletes at least one edge from L_G)
- At most $|V|$ retreats per phase. $\longleftarrow O(|E|+|V|)$ per phase (because a retreat deletes one node from L_G)
- At most $|E||V|$ advances per phase. $\longleftarrow O(|E||V|)$ per phase

Dinitz' Algorithm: Analysis

Lemma

A phase can be implemented to run in $O(|E||V|)$ time.

Proof.

- Initialization happens once per phase. using BFS
- At most $|E|$ augmentations per phase. $\longleftarrow O(|E|)$ per phase (because an augmentation deletes at least one edge from L_G)
- At most $|V|$ retreats per phase. $\longleftarrow O(|E| + |V|)$ per phase (because a retreat deletes one node from L_G)
- At most $|E||V|$ advances per phase. $\longleftarrow O(|E||V|)$ per phase (because at most $|V|$ advances before retreat or augmentation)

Dinitz' Algorithm: Analysis

Theorem (Dinitz 1970)

Dinitz' algorithm runs in $O(|E||V|^2)$ *time.*

Dinitz' Algorithm: Analysis

Theorem (Dinitz 1970)

Dinitz' algorithm runs in $O(|E||V|^2)$ time.

Proof.

Dinitz' Algorithm: Analysis

Theorem (Dinitz 1970)

Dinitz' algorithm runs in $O(|E||V|^2)$ time.

Proof.

- By Lemma, $O(|E||V|)$ time per phase.

Dinitz' Algorithm: Analysis

Theorem (Dinitz 1970)

Dinitz' algorithm runs in $O(|E||V|^2)$ time.

Proof.

- By Lemma, $O(|E||V|)$ time per phase.
- At most $|V| - 1$ phases

Dinitz' Algorithm: Analysis

Theorem (Dinitz 1970)

Dinitz' algorithm runs in $O(|E||V|^2)$ time.

Proof.

- By Lemma, $O(|E||V|)$ time per phase.
- At most $|V| - 1$ phases (as in shortest-augmenting-path analysis).

Summary

year	method	# augmentations	running time
1955	augmenting path	$\lvert V\rvert C$	$O(\lvert E\rvert \lvert V\rvert C)$
1972	fattest path	$\lvert E\rvert \log(\lvert E\rvert C)$	$O\left(\lvert E\rvert^2 \log n \log(\lvert E\rvert C)\right)$
1972	capacity scaling	$\lvert E\rvert \log C$	$O\left(\lvert E\rvert^2 \log C\right)$
1985	improved capacity scaling	$\lvert E\rvert \log C$	$O(\lvert E\rvert \lvert V\rvert \log C)$
1970	shortest augmenting path	$\lvert E\rvert \lvert V\rvert$	$O\left(\lvert E\rvert^2 \lvert V\rvert\right)$
1970	level graph	$\lvert E\rvert \lvert V\rvert$	$O\left(\lvert E\rvert \lvert V\rvert^2\right)$
1983	dynamic trees	$\lvert E\rvert \lvert V\rvert$	$O(\lvert E\rvert \lvert V\rvert \log \lvert V\rvert)$

augmenting-path algorithms with integer capacities between 1 and C

Theory Highlights

year	method	worst case	discovered by
1951	simplex	$O\left(\lvert E\rvert\lvert V\rvert^2 C\right)$	Dantzig
1955	augmenting paths	$O(\lvert E\rvert\lvert V\rvert C)$	Ford–Fulkerson
1970	shortest augmenting paths	$O\left(\lvert E\rvert\lvert V\rvert^2\right)$	Edmonds–Karp, Dinitz
1974	blocking flows	$O\left(\lvert V\rvert^3\right)$	Karzanov
1983	dynamic trees	$O(\lvert E\rvert\lvert V\rvert\log n)$	Sleator–Tarjan
1985	improved capacity scaling	$O(\lvert E\rvert\lvert V\rvert\log C)$	Gabow
1988	push–relabel	$O\left(\lvert E\rvert\lvert V\rvert\log\left(\lvert V\rvert^2/\lvert E\rvert\right)\right)$	Goldberg–Tarjan
1998	binary blocking flows	$O\left(\lvert E\rvert^{3/2}\log\left(n^2/\lvert E\rvert\right)\log C\right)$	Goldberg–Rao
2013	compact networks	$O(\lvert E\rvert\lvert V\rvert)$	Orlin
2014	interior-point methods	$\tilde{O}\left(\lvert E\rvert\lvert E\rvert^{1/2}\log C\right)$	Lee–Sidford
2016	electrical flows	$\tilde{O}\left(\lvert E\rvert^{10/7}C^{1/7}\right)$	Madry
20xx		???	

augmenting-path algorithms with integer capacities between 1 and C

Maximum-Flow: Practice

Push–relabel algorithm (Section 7.4) of [KT05]. [Goldberg–Tarjan 1988]

Increases flow one edge at a time instead of one augmenting path at a time.

Maximum-Flow: Practice

Caveat. Worst-case running time is generally not useful for predicting or comparing max-flow algorithm performance in practice.

Best in practice. Push–relabel method with gap relabeling: $O(|E|^{3/2})$ in practice.

Referred Materials

- Content of this lecture comes from Section 7.3 in [KT05].
- Suggest to read Chapter 26 in [CLRS09].