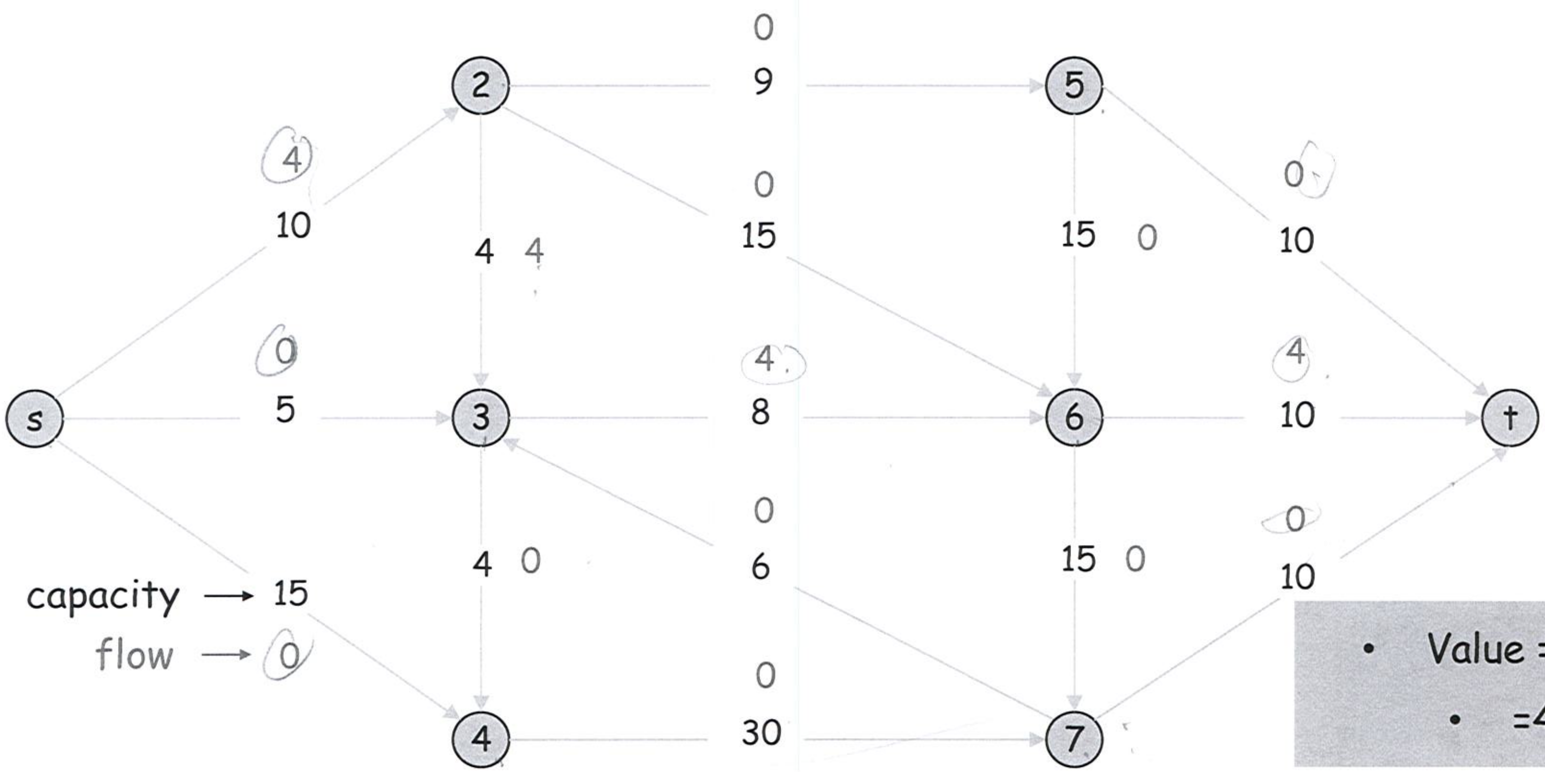
Flows

Def. An s-t flow is a function that satisfies:

- For each $e \in E$: $0 \leq f(e) \leq c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [conservation]

Def. The value of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$. $= \sum_{e \text{ in } t} f(e)$



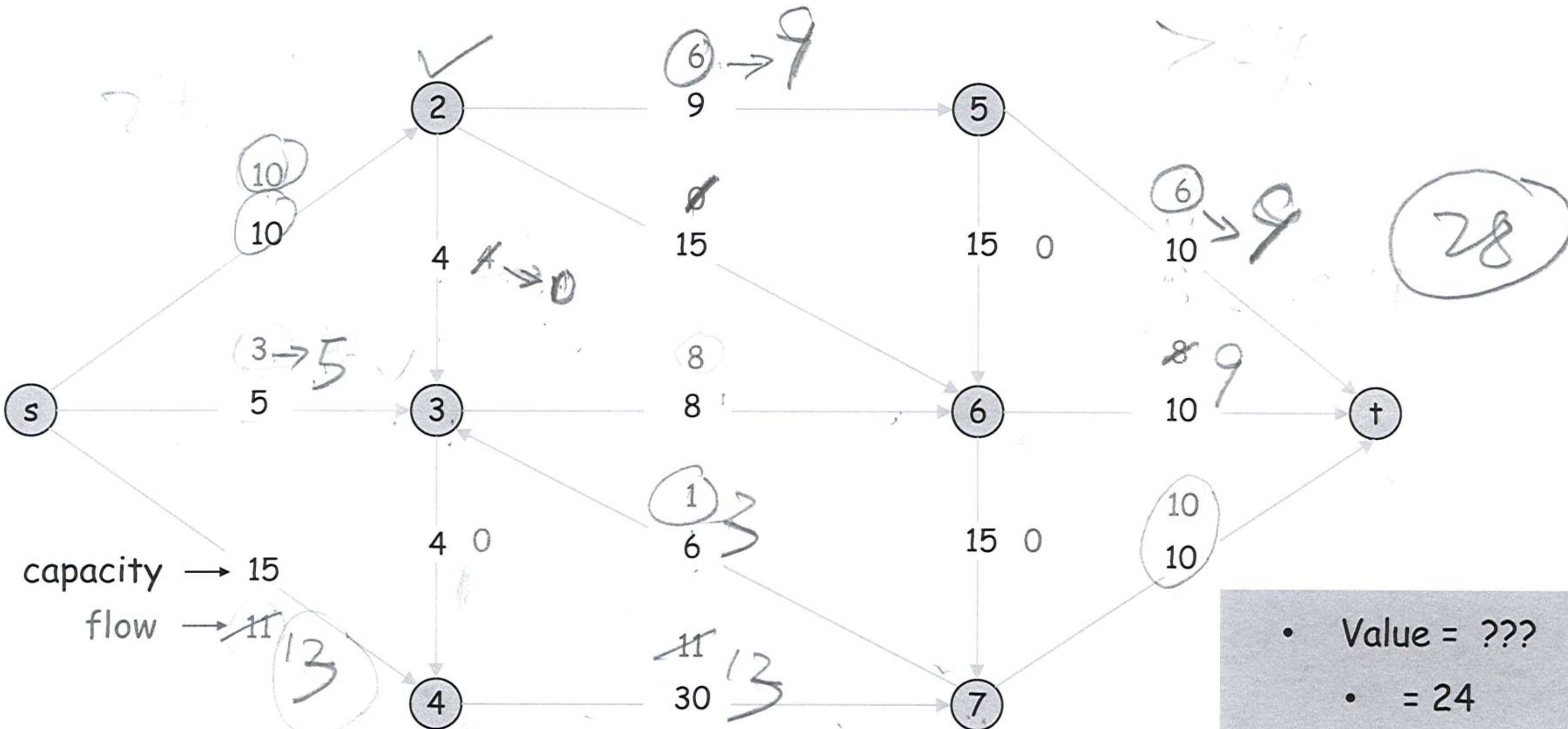
- Value = ???
  - =4

Flows

Def. An **s-t flow** is a function that satisfies:

- For each $e \in E$: $0 \le f(e) \le c(e)$ [capacity]
- For each $v \in V - \{s, t\}$: $\sum_{e \text{ in to } v} f(e) = \sum_{e \text{ out of } v} f(e)$ [conservation]

Def. The **value** of a flow f is: $v(f) = \sum_{e \text{ out of } s} f(e)$.



- Value = ???
- = 24

Towards a Max Flow Algorithm

Can not change decision

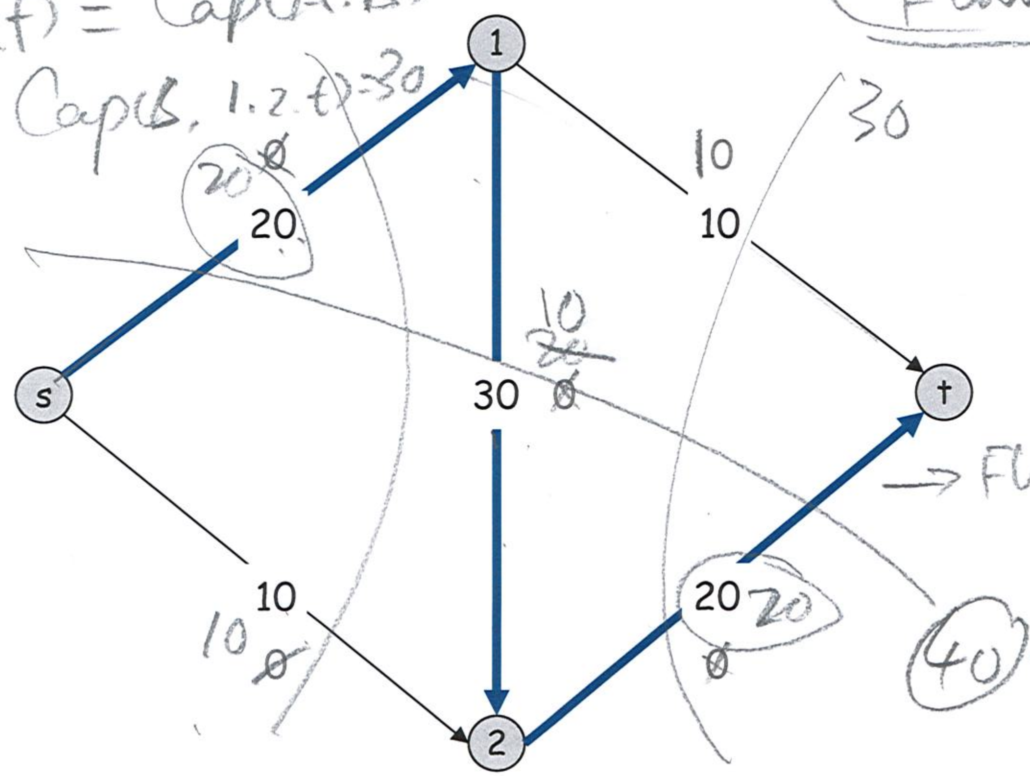
Greedy algorithm.

- Start with $f(e) = 0$ for all edge $e \in E$.
- Find an s-t path P where each edge has $f(e) < c(e)$.
- Augment flow along path P.
- Repeat until you get stuck.

Val(f) = Cap(A.B)

Cap(B, 1.2.t) = 30

Flow = 20 OPT ?



→ Flow = 30. ← OPT ?!

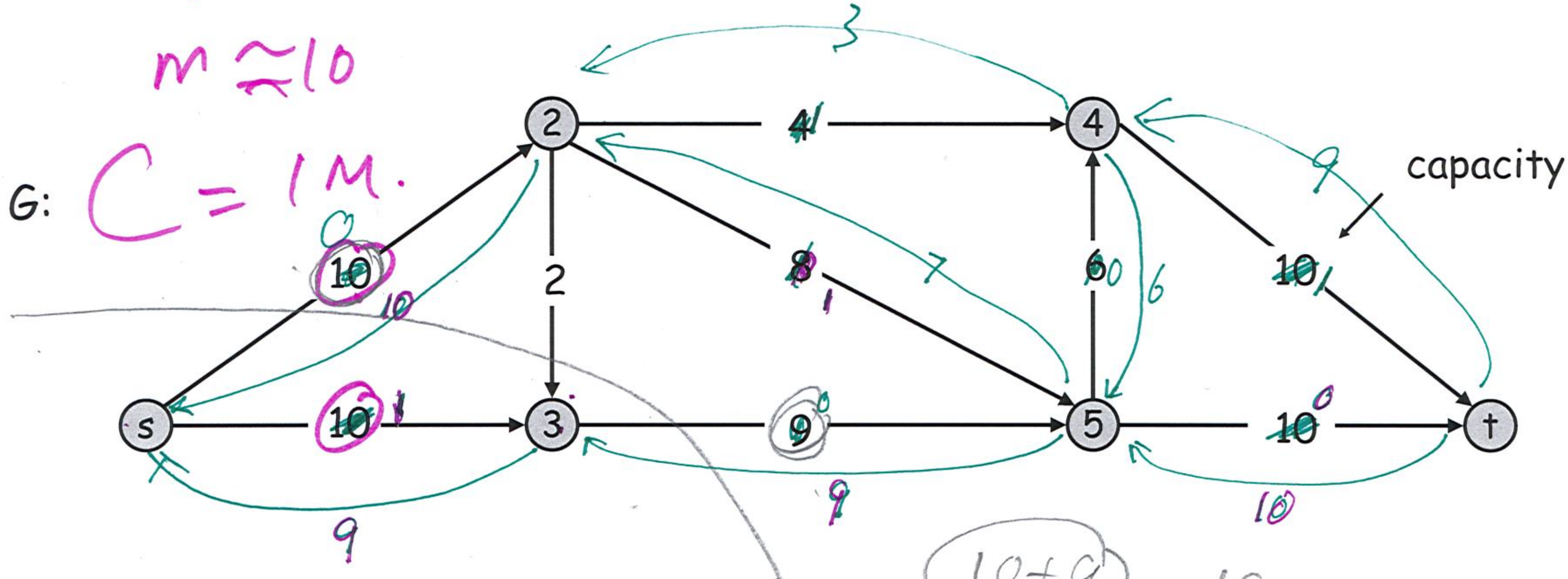
Flow value = ~~0~~ 20

Ford-Fulkerson Algorithm

$n=6$

$m \approx 10$

$C = 1M.$



$(10+9) = 19$

$C = 10+10 = 20.$

$Cap(\{s,3\}, \{2,4,5,t\})$

$f \to 20.$

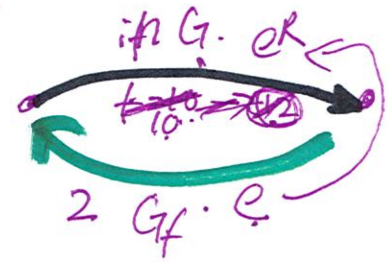
$C = 20.$

Augmenting Path Algorithm

$O(mC)$ $m \approx n^2$ value of input flow Pseudo-Poly

f is a flow function that maps each edge e to a nonnegative number: E → R+
f(e): amount of flow carried by edge e

$b=2?$




```
O(m) Augment(f, c, P) {
        b ← bottleneck(P)      Minimum Capacity on P.
        foreach e ∈ P {
G.         if (e ∈ E) f(e) ← f(e) + b
           else       f(eR) ← f(eR) - b
        }
        return f
     }
```

forward edge

reverse edge

$C/1 = C = \sum_{e \text{ out of } s} c(e)$

$e \geq 1$

$f = f + b$

input   Capacity

```
Ford-Fulkerson(G, s, t, c) {
    foreach e ∈ E  f(e) ← 0
    Gf ← residual graph
                                               s→t
C? ← while (there exists augmenting path P) {
    O(m) f ← Augment(f, c, P)                in Gf.
    O(m) update Gf
    }
    return f ← Flow function.                 f(e) → number
}          max flow.
```

Gf. → forward edge in G. / reverse edge $f(e^R) = f(e^R) - b$

? DFS / BFS → OR. Find Path

Ford-Fulkerson: Exponential Number of Augmentations

Q. Is generic Ford-Fulkerson algorithm polynomial in input size?

m, n, and log C

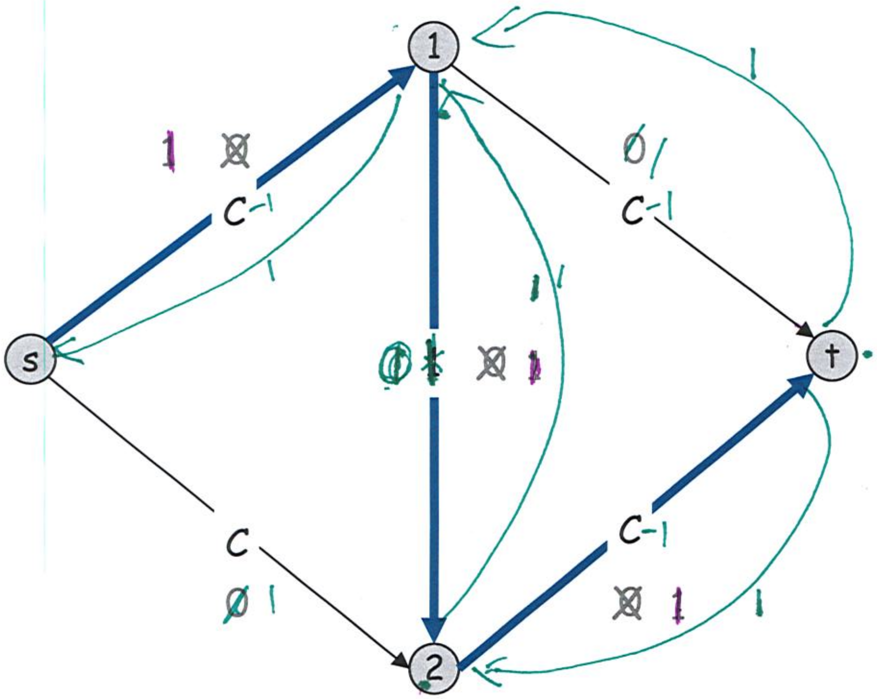
A. No. If max capacity is C, then algorithm can take C iterations.

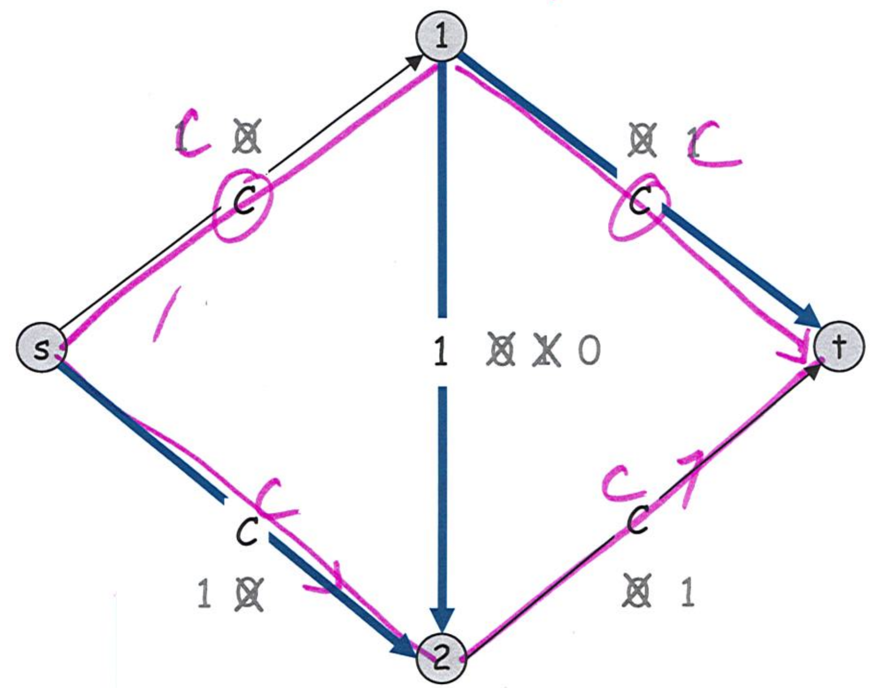
2C*

shortest path.

Largest bp.

f = 2C.





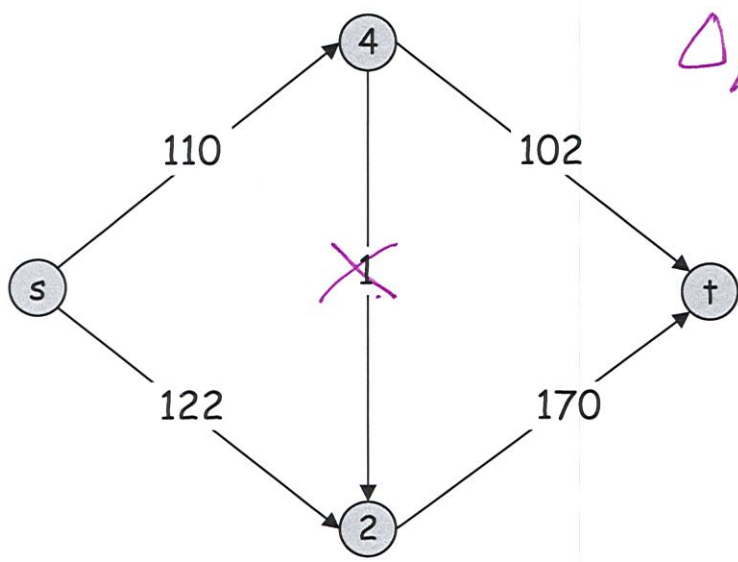
Capacity Scaling

Intuition. Choosing path with highest bottleneck capacity increases flow by max possible amount.

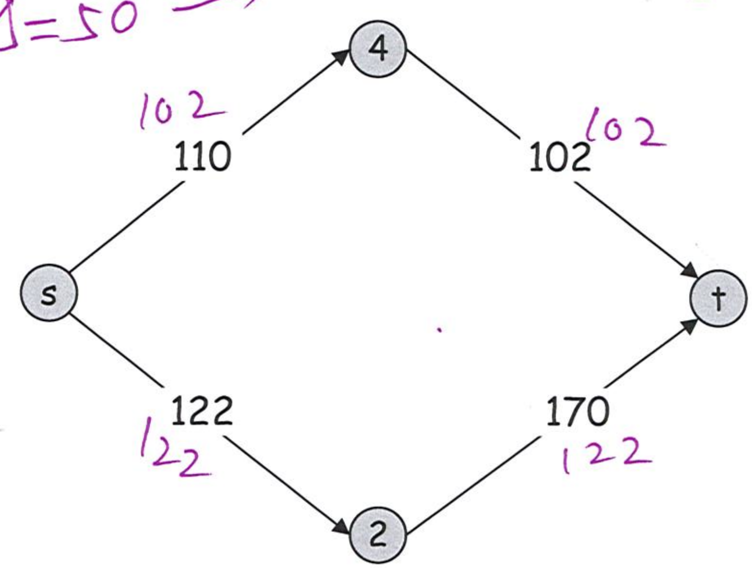
- Don't worry about finding exact highest bottleneck path.
- Maintain scaling parameter Δ.
- Let $G_f(\Delta)$ be the subgraph of the residual graph consisting of only arcs with capacity at least Δ.

$\Delta = 100$

$\Delta/2$. $\Delta = 50 \rightarrow 25 \rightarrow 12 \rightarrow 6 \rightarrow 3 \rightarrow 1$.



G_f



$G_f(100)$

Capacity Scaling

$m^2 \log_2 C$

C big C → logC.

```
Scaling-Max-Flow(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   Δ ← largest power of 2 no greater than C
   Gf ← residual graph

   while (Δ ≥ 1) {
      Gf(Δ) ← Δ-residual graph
      while (there exists augmenting path P in Gf(Δ)) {
         f ← augment(f, c, P)
         update Gf(Δ)
      }
      Δ ← Δ / 2
   }
   return f
}
```

128 → 64 → 32 → 16 → 8 → 4 → 2 → 1

← Keep Reduce Δ

Remove edge with $c(e) < \Delta$

$1 + \lceil \log_2 C \rceil$ times

<= 2m times (Theorem 7.19)

O(m)

O(m)

$\Delta \le 1$