We have a list A of n integers, for some $n = 2^k - 1$, each written in binary. Every number in the range 0 to n is in the list exactly once, except for one. However, we cannot directly access the value of integer $A[i]$ (for any i); instead, we can only access the j^{th} bit of i: $A[i][j]$. Our goal is to find the missing number.
Give an algorithm to find the missing integer that uses $\Theta(n)$ bit accesses.
Note: a solution that accesses *every* bit will take time $\Theta(n \log n) = \Theta(nk)$ and will receive no credit.

$K = 3$

$$\left.\begin{array}{l}0\ 0\ 0\\0\ 0\ 1\\0\ 1\ 1\end{array}\right\} 3\times 0$$

$$\left.\begin{array}{l}1\ 0\ 0\\1\ 0\ 1\\1\ 1\ 0\\1\ 1\ 1\end{array}\right\} 4\times 1$$

0 0 0

0 0 1

0 1 1

0 1 0

Warm-Up (Solution)

To find MSB: count leading 0s & 1s
whichever has fewer:
- that is missing MSB
- Keep only those w/ this bit
recurse on next bit on remainder

$$T(n) = n + T\left(\frac{n}{2}\right) = n + \frac{n}{2} + \frac{n}{4} + \ldots 1$$
$$= n\left(1 + \frac{1}{2} + \frac{1}{4} + \ldots \frac{1}{2^k}\right)$$

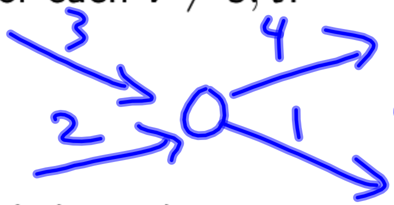
Definition An $s - t$ flow puts f_e flow on each edge e such that:

- Each capacity is obeyed; this is called the "capacity constraint"

$$0 \leq f_e \leq c_e$$

- "Conservation of flow": only the source can create flow, and only the sink can consume it. More formally, for each $v \neq s, t$:

$$\sum_{e \text{ into } v} f_e = \sum_{e \text{ out of } v} f_e$$



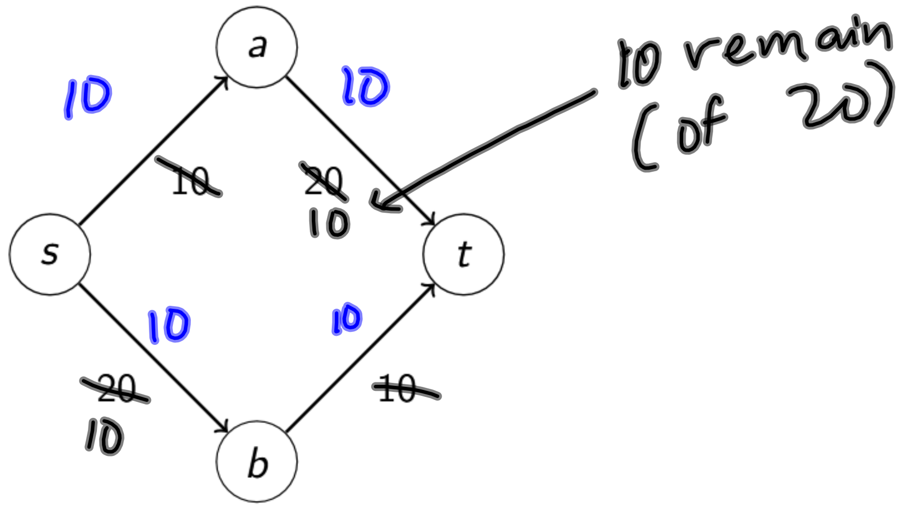
ok!

- The source can create any amount of flow, and the sink can consume any amount. We say the *value* of a given flow f is:

$$v(f) = \sum_{e \text{ out of } s} f_e$$

blue: flow

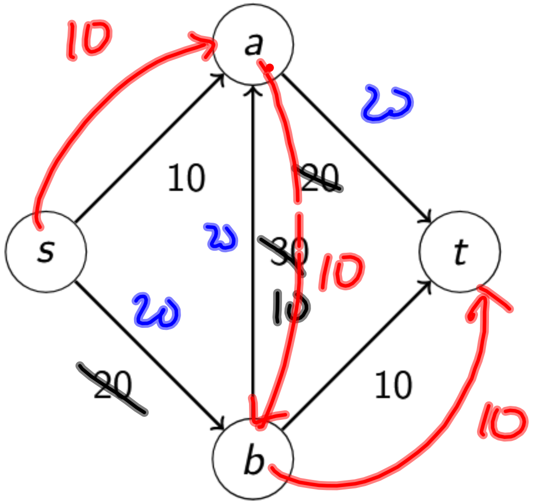
Introduction

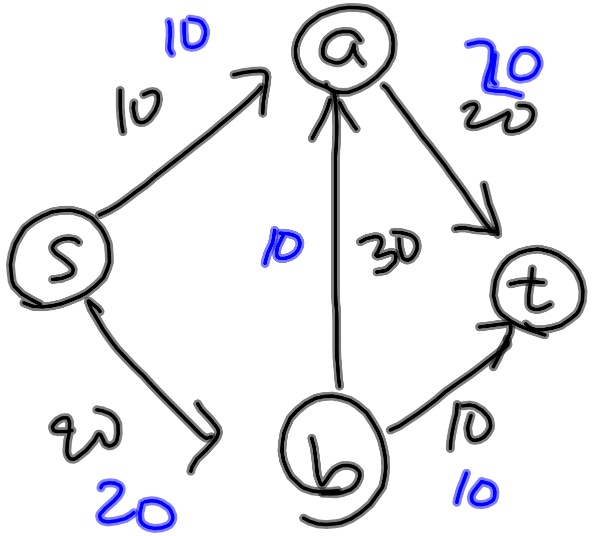


- Does the greedy algorithm in the handout work here?

Sure, on this graph . . .

And with another graph...

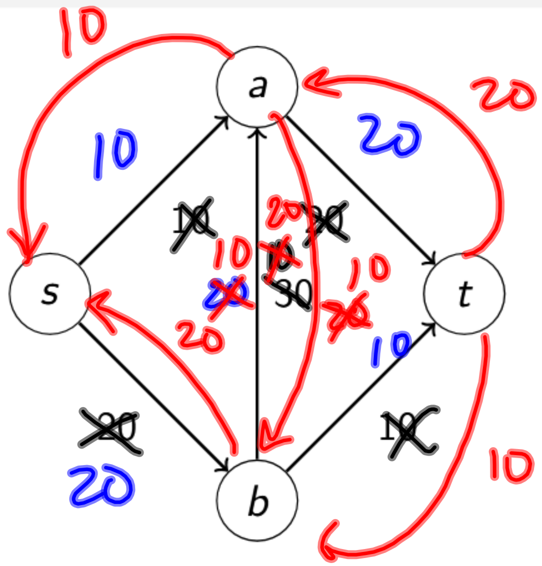


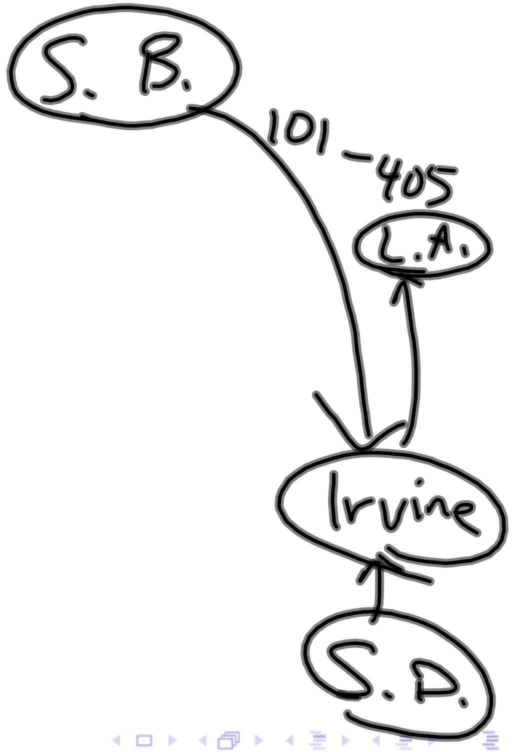


back edges

Can we "take back" flow?

Residual Graph G_f





Recall that all c_e values are positive integers. Will any f_e values ever be non-integer? Why or why not?

No. (f_e always int)

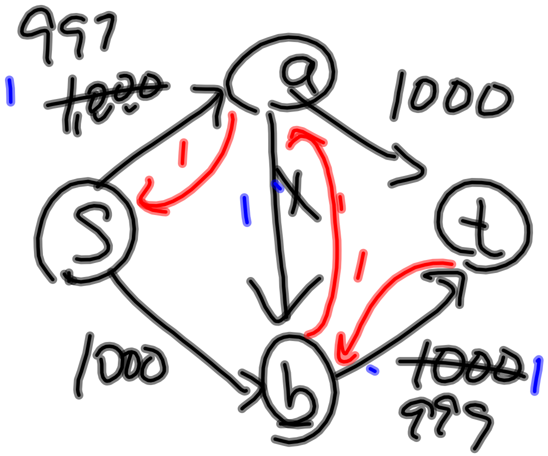
What is the minimum increase in $v(f)$ each iteration of the while loop?

Yes, at least one

Define $C = \sum c_e$ over all $e = (s, v)$. How many iterations will the **while** loop take in the worst case? Can you give an input that will take that long?

while $\exists$ path p from s to t in G_f **do**
  // Increase $v(f)$ by at least one

O(c) iter (worst case)



s-a-b-t
s-b-a-t
s-a-b-t
s-b-a-t

1000 times

How long does each iteration of the **while** loop take?

while $\exists$ path p from s to t in G_f **do**
  p = any simple s to t path in G_f. ← $O(m)$
  b = min residual capacity edge on p (the "bottleneck" edge)
  for all edges $e = (u, v)$ in p **do**
    if e is forward **then**
      $f_e = f_e + b$
    else
      $e' = (v, u)$
      $f_{e'} = f_{e'} - b$

$O(m)$ total

What is the total running time of the algorithm?

$O(C)$

$\forall_e \quad f_e = 0$
while $\exists$ path p from s to t in G_f **do**
  $p =$ any simple s to t path in G_f.
  $b =$ min residual capacity edge on p (the "bottleneck" edge)
  for all edges $e = (u, v)$ in p **do**
    if e is forward **then**
      $f_e = f_e + b$
    else
      $e' = (v, u)$
      $f_{e'} = f_{e'} - b$

$\frac{O(m)}{\text{iter.}}$

$O(mC)$

Is this polynomial time?

C is integer VALUE

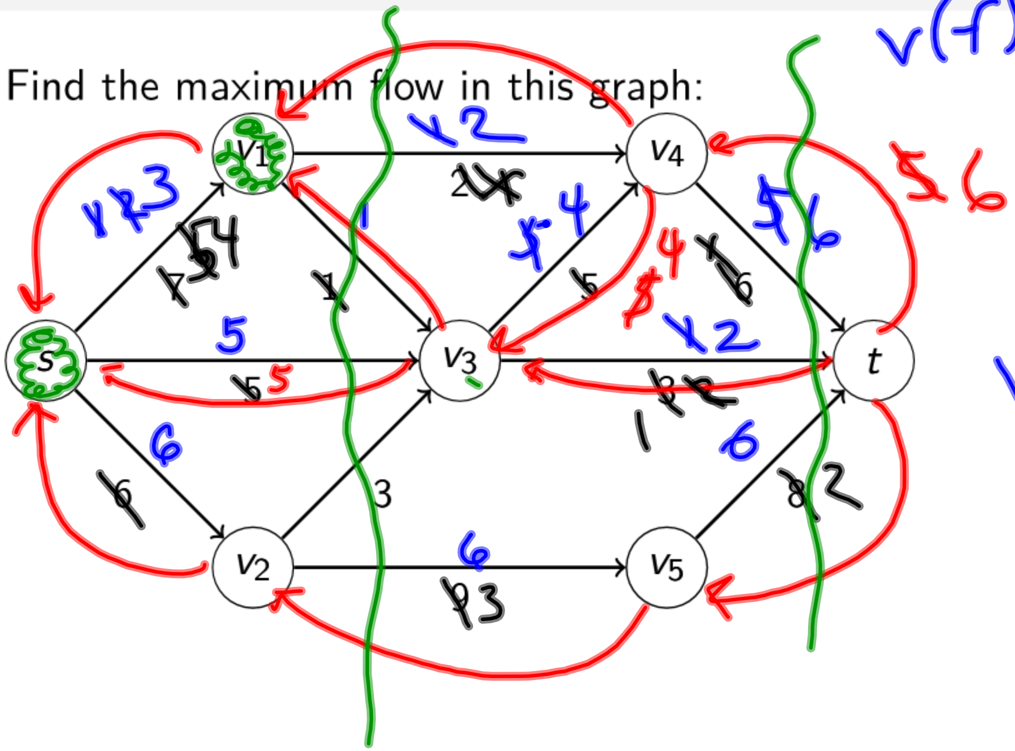
so no.

$O(m \underbrace{\log C}_{\text{size of input}})$ would be poly

Review

Find the maximum flow in this graph:

$v(f) = 14$



$v(f) \le \sum c_e$

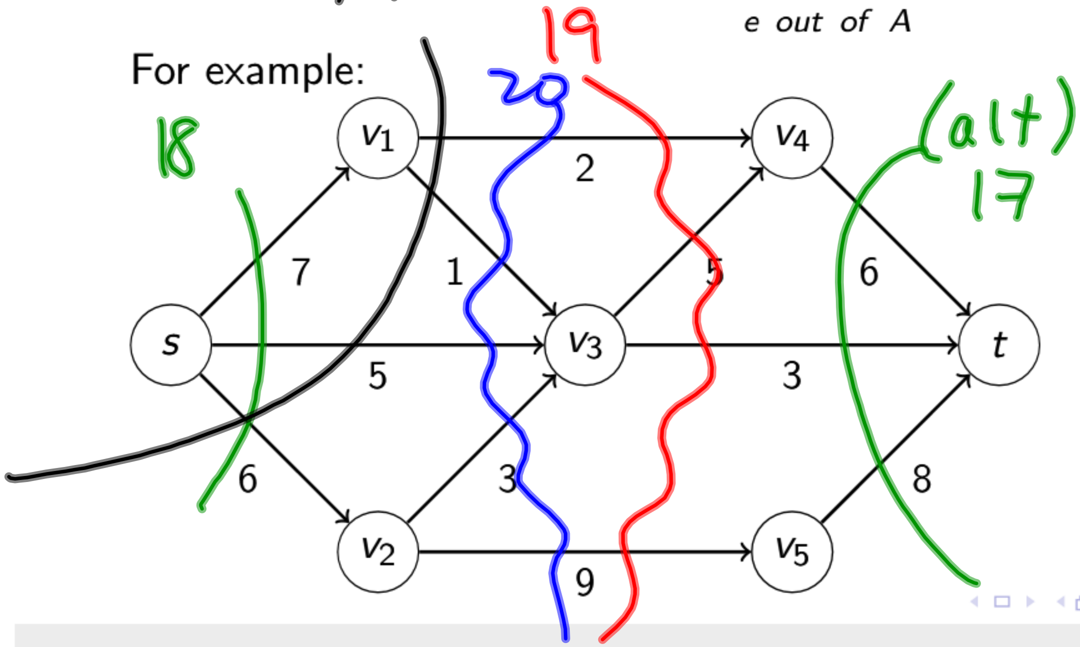
e out of s

Define (A,B) cut: a partition of V into sets A and B such that $s \in A$, $t \in B$. We say that the *capacity* of the cut is:

A^*/B^*

$$c(A,B) = \sum_{e \text{ out of } A} c_e$$

For example:



Claim: $v(f) = f^{out}(A) - f^{in}(A)$

$$v(f) = f^{out}(s) - 0$$

$$= f^{out}(s) - f^{in}(s)$$

// $\forall v \in A$, except s, $f^{in}(v) = f^{out}(v) \longrightarrow 0 = f^{out}(v) - f^{in}(v)$

$$+ \sum_{\substack{v \in A \\ \text{exc. } s}} f^{out}(v) - f^{in}(v)$$

$$= \sum_{v \in A} f^{out}(v) - f^{in}(v)$$

Claim: $v(f) = f^{in}(B) - f^{out}(B)$

- How can I prove this?

Mirror of last

$$f^{out}(A) = f^{in}(B)$$

$$f^{in}(A) = f^{out}(B)$$

Proof of Correctness

Claim: $v(f) \leq c(A, B)$

Pf: $v(f) = f^{out}(A) - f^{in}(A)$

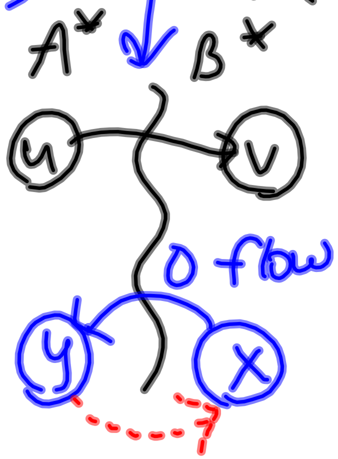
$\leq f^{out}(A)$ ← eq

$\leq \sum_{e \text{ out of } A} f_e \leq \sum_{e \text{ out of } A} c_e = c(A, B)$

Claim: If there is no $s - t$ path in G_f, then there is a cut (A^*, B^*) such that $v(f) = c(A^*, B^*)$

$$A^* = \{v \mid s \rightsquigarrow v \text{ in } G_f\}$$

must be saturated



$$v(f) = f^{out}(A^*) - f^{in}(A^*)$$

$$= \sum_{e \text{ out } A^*} f_e - \sum_{e \text{ in } A^*} f_e$$

$$= \sum_{e \text{ out of } A^*} C_e - \sum_{e \text{ in } A^*} 0 = \sum_{e \text{ out of } A^*} C_e = c(A^*, B^*)$$

Suppose you are given a directed graph $G = (V, E)$, with a positive integer capacity c_e on each edge e, a designated source $s \in V$, and a designated sink $t \in V$. You are also given an integer maximum $s - t$ flow in G, defined by a flow value f_e on each edge e.

Now suppose we pick a specific edge $e \in E$ and increase its capacity by one unit. Show how to find a maximum flow in the resulting capacitated graph in time $O(m + n)$.

- form G_f $O(m)$
- inc. c_e in G_f
- if $s \rightsquigarrow t$, augment
- else no change

What if it had been a decrease of one instead?

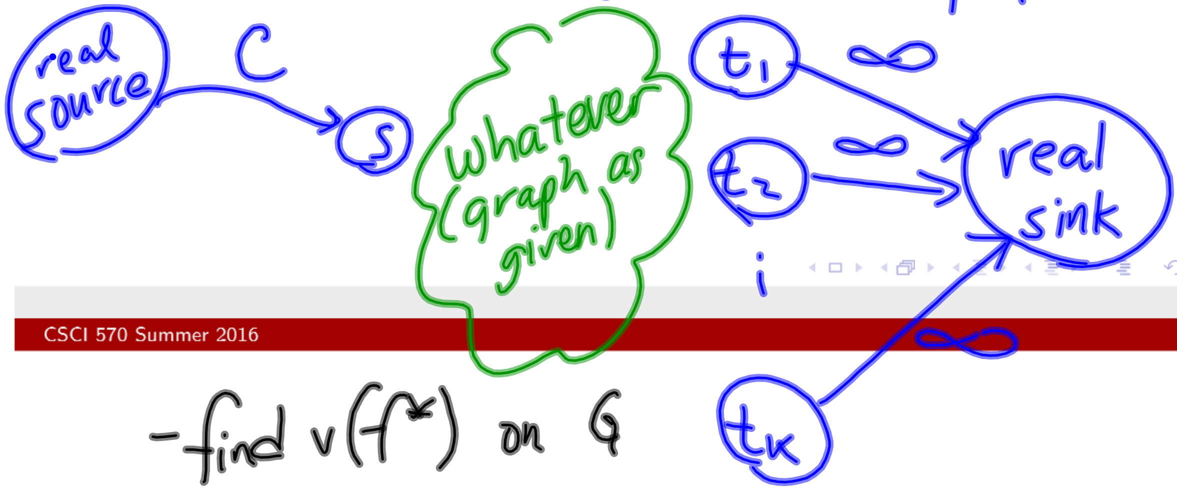
if wasn't saturated, no change

$$s \rightsquigarrow u \xrightarrow{e} v \rightsquigarrow t$$

- find paths $s \rightsquigarrow u$, $v \rightsquigarrow t$ in f
- dec f_e on them by one
- at most one augment to go...

Suppose a concert has just ended and C cars are parked at the venue. We would like to determine how long it takes for all of them to leave the area. For this problem, we are given a graph representing the road network where all cars start at a particular vertex s (the parking lot) and several vertices ($t_1, t_2, \ldots, t_k$) are designated as exits. We are also given capacities (in cars per minute) for each road (directed edges). Give a polynomial-time algorithm to determine the amount of time necessary to get all cars out of the area.

assume have poly time flow alg



- find $v(f^*)$ on G
- $v(f^*)$ is max exit per minute

add src, sink

Given G, X, and S, show how to decide in polynomial time whether such a set of evacuation routes exists.

for all $v \in X$
add (src, v) cap 1

for all safe node s
add (s, sink) cap ∞

$\forall e$ $c_e = 1$ (except ↑)
find $v(f^*)$ // find max flow
if $v(f^*) = |X|$, done.
else not possible