

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100

# Algorithms

Country	Percentage
United States	100%
Canada	100%
United Kingdom	100%
France	100%
Germany	100%
Italy	100%
Spain	100%
Japan	100%
China	100%
India	100%
Brazil	100%
South Africa	100%
Australia	100%
Sweden	100%
Norway	100%
Denmark	100%
Finland	100%
Poland	100%
Czech Republic	100%
Slovak Republic	100%
Hungary	100%
Slovenia	100%
Croatia	100%
Serbia	100%
Bulgaria	100%
Romania	100%
Greece	100%
Turkey	100%
Israel	100%
Ukraine	100%
Belarus	100%
Belgium	100%
Netherlands	100%
Switzerland	100%
Austria	100%
Luxembourg	100%
Portugal	100%
Ireland	100%
Malta	100%
Cyprus	100%
Latvia	100%
Lithuania	100%
Estonia	100%
Malaysia	100%
Singapore	100%
Philippines	100%
Indonesia	100%
Thailand	100%
Vietnam	100%
Myanmar	100%
South Korea	100%
North Korea	100%
Japan	100%
China	100%
India	100%
Brazil	100%
South Africa	100%
Australia	100%
Sweden	100%
Norway	100%
Denmark	100%
Finland	100%
Poland	100%
Czech Republic	100%
Slovak Republic	100%
Hungary	100%
Slovenia	100%
Croatia	100%
Serbia	100%
Bulgaria	100%
Romania	100%
Greece	100%
Turkey	100%
Israel	100%
Ukraine	100%
Belarus	100%
Belgium	100%
Netherlands	100%
Switzerland	100%
Austria	100%
Luxembourg	100%
Portugal	100%
Ireland	100%
Malta	100%
Cyprus	100%
Latvia	100%
Lithuania	100%
Estonia	100%
Malaysia	100%
Singapore	100%
Philippines	100%
Indonesia	100%
Thailand	100%
Vietnam	100%
Myanmar	100%
South Korea	100%
North Korea	100%
Japan	100%
China	100%
India	100%
Brazil	100%
South Africa	100%
Australia	100%
Sweden	100%
Norway	100%
Denmark	100%
Finland	100%
Poland	100%
Czech Republic	100%
Slovak Republic	100%
Hungary	100%
Slovenia	100%
Croatia	100%
Serbia	100%
Bulgaria	100%
Romania	100%
Greece	100%
Turkey	100%
Israel	100%
Ukraine	100%
Belarus	100%
Belgium	100%
Netherlands	100%
Switzerland	100%
Austria	100%
Luxembourg	100%
Portugal	100%
Ireland	100%
Malta	100%
Cyprus	100%
Latvia	100%
Lithuania	100%
Estonia	100%
Malaysia	100%
Singapore	100%
Philippines	100%
Indonesia	100%
Thailand	100%
Vietnam	100%
Myanmar	100%
South Korea	100%
North Korea	100%
Japan	100%
China	100%
India	100%
Brazil	100%
South Africa	100%
Australia	100%
Sweden	100%
Norway	100%
Denmark	100%
Finland	100%
Poland	100%
Czech Republic	100%
Slovak Republic	100%
Hungary	100%
Slovenia	100%
Croatia	100%
Serbia	100%
Bulgaria	100%
Romania	100%
Greece	100%
Turkey	100%
Israel	100%
Ukraine	100%
Belarus	100%
Belgium	100%
Netherlands	100%
Switzerland	100%
Austria	100%
Luxembourg	100%
Portugal	100%
Ireland	100%
Malta	100%
Cyprus	100%
Latvia	100%
Lithuania	100%
Estonia	100%
Malaysia	100%
Singapore	100%
Philippines	100%
Indonesia	100%
Thailand	100%
Vietnam	100%
Myanmar	100%
South Korea	100%
North Korea	100%
Japan	100%
China	100%
India	100%
Brazil	100%
South Africa	100%
Australia	100%
Sweden	100%
Norway	100%
Denmark	100%
Finland	100%
Poland	100%
Czech Republic	100%
Slovak Republic	100%
Hungary	100%
Slovenia	100%
Croatia	100%
S	

# NETWORK FLOWS

Max-Flow

Initially  $f(e) = 0$  for all  $e$  in  $G$

While there is an s-t path in the residual graph  $G_f$

Let  $P$  be a simple s-t path in  $G_f$

$f' = \text{augment}(f, P)$

Update  $f$  to  $f'$

Update the residual graph  $G_f$  to be  $G_{f'}$

EndWhile

Return  $f$

LOOP OVER PATHS

(CALLED AUGMENTING  
PATHS)

$\text{augment}(f, P)$ :

Let  $b = \text{bottleneck}(P, f)$

For each edge  $(u, v) \in P$

If  $e = (u, v)$  is a forward edge then

increase  $f(e)$  in  $G$  by  $b$

Else  $((u, v)$  is a backward edge, and let  $e = \begin{pmatrix} (v, u) \\ \underline{(u, v)} \end{pmatrix}$ )

decrease  $f(e)$  in  $G$  by  $b$

Endif

Endfor

Return( $f$ )

**\*\*Here we are constructing the flow. So for backward edges  $u \rightarrow v$ , we need to decrease the flow of  $v \rightarrow u$  by  $b$ .**

# NETWORK FLOWS

## Ford Fulkerson Algorithm.

Repeatedly find a path from  $s$  to  $t$  in which all edges have some leftover capacity and send flow equal to the capacity of the *bottleneck edge*.

*bottleneck edge: edge with minimum capacity in the augmenting path*

- Requires that all capacities are integers.
- Running time:  $O((m + n) \cdot f)$ , where  $f$  is the value of the flow returned.

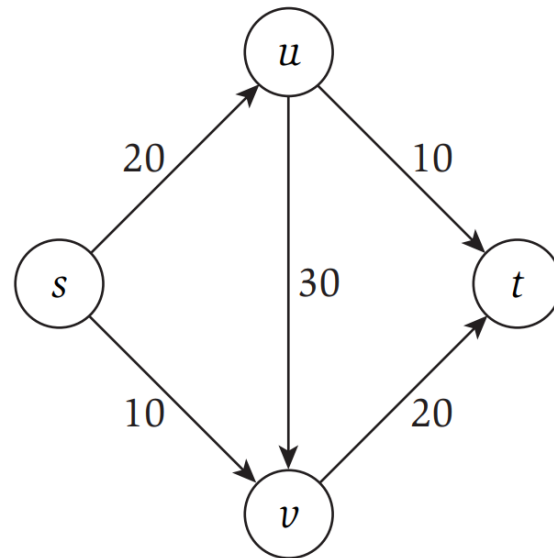
## SHORTEST PATH AUGMENTATION

### Edmonds-Karp Algorithm.

Always use a shortest  $s \rightsquigarrow t$  path (i.e. a path with the minimum no. of edges) in which all edges have non-zero residual capacity as the augmenting path.

The amount of flow sent along the augmenting path is equal to the capacity of a *bottleneck* edge i.e. an edge with the minimum capacity in the path.

**HOW MANY AUGMENTATIONS WILL THE ALGORITHM DO? (TOTAL NUMBER OF LOOPS)**



## SHORTEST PATH AUGMENTATION

### Edmonds-Karp Algorithm.

Always use a shortest  $s \rightsquigarrow t$  path (i.e. a path with the minimum no. of edges) in which all edges have non-zero residual capacity as the augmenting path.

The amount of flow sent along the augmenting path is equal to the capacity of a *bottleneck* edge i.e. an edge with the minimum capacity in the path.

**Claim.** There are at most  $mn$  augmentations.  $n = \#vertices, m = \# edges$

The augmenting path at any stage can be found in  $O(m + n)$  time using BFS.

Thus, the overall running time is:  $O((m + n) \cdot mn)$ .

### DEFINITION.

For any vertex  $v$ , let  $d(v)$  denote the length of the shortest  $s \rightsquigarrow v$  path in the *current* residual graph in which each edge has non-zero residual capacity.

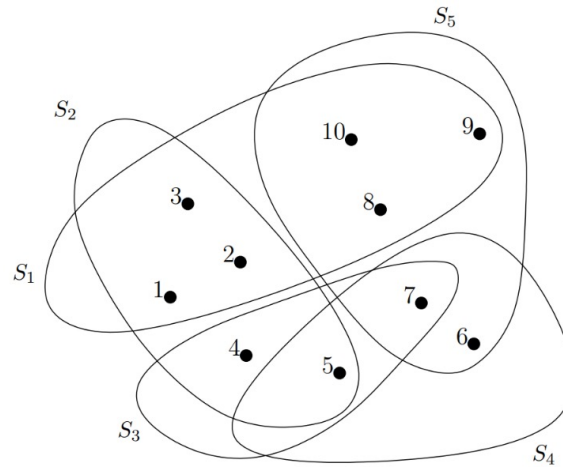
# SET COVER

**Input.** A set  $X$  with  $n$  elements called the *ground set*.

Subsets:  $S_1, S_2, \dots, S_m \subseteq X$ .

**Goal.** Select the smallest number of sets in  $\{S_1, \dots, S_m\}$  whose union is  $X$ .

*Example.*



$X = \{1, 2, \dots, 10\}.$

*Sets :*  $S_1, \dots, S_5$

*The smallest collection of these sets that “covers” all elements is  $\{S_2, S_5\}$ .*

## SET COVER

**Input.** A set  $X$  with  $n$  elements called the *ground set*.

Subsets:  $S_1, S_2, \dots, S_m \subseteq X$ .

**Goal.** Select the smallest number of sets in  $\{S_1, \dots, S_m\}$  whose union is  $X$ .

HOW COMPLEX IS A BRUTE FORCE APPROACH?

$O(2^m)$

TRY A GREEDY APPROACH TO SOLVE THE PROBLEM

DOES YOUR APPROACH ACHIEVE AN OPTIMAL SOLUTION?

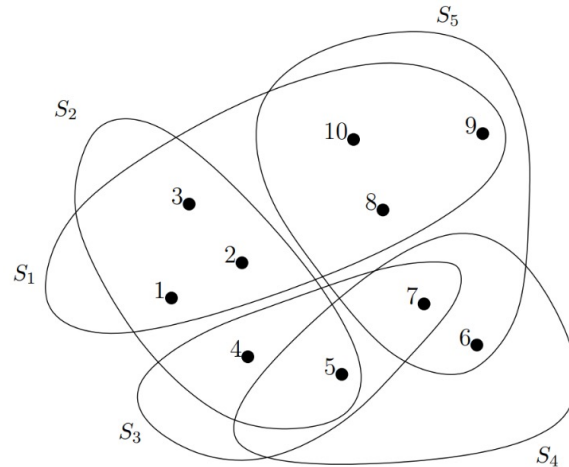
# SET COVER

## Greedy Algorithm.

We select sets “greedily” one by one.

We always pick the set that covers the maximum number of new elements (i.e., elements not covered by previously picked sets).

What answer do we get for the following input?



1<sup>st</sup> set:  $S_1$  (6 new elements)

2<sup>nd</sup> set:  $S_3$  (3 new elements)

3<sup>rd</sup> set:  $S_4$  (1 new element)

*So this algorithm does not give an optimal answer!*



# SET COVER

## Greedy Algorithm.

We select sets “greedily” one by one.

We always pick the set that covers the maximum number of new elements (i.e., elements not covered by previously picked sets).

THE SET COVER PROBLEM IS AN **NP-HARD PROBLEM**

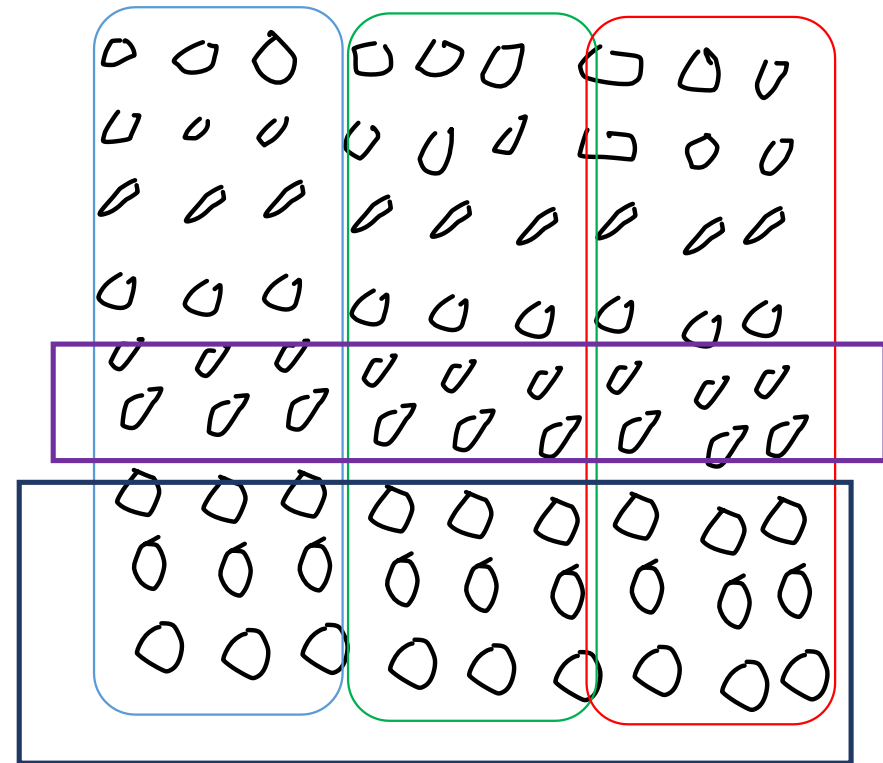
SO WE HAVE TO COMPROMISE ON SOMETHING IF WE WANT TO SOLVE IT IN  
**POLYNOMIAL TIME**

**WE THIS APPROACH WE ARE COMPROMISING ON CORRECTNESS**

**BUT HOW GOOD IS THE SOLUTION?**

# SET COVER

WHAT COULD THE GREEDY ALGORITHM OUTPUT?  
(USE TIE BREAKING FOR SIMILAR CASES)



## SET COVER

**Theorem.** The greedy algorithm always returns an answer whose size is at most  $\ln n$  times that of the optimal solution.  $n = |X|$

*Proof.* Suppose that the optimal solution has  $k$  sets.

Let  $n_t$  be the number of uncovered elements in the  $t^{\text{th}}$  iteration.

Since all elements are uncovered initially  $\frac{n_1}{n_0} = n$ .

Can be proven by  
induction on  $n_t$

Since the uncovered elements are covered by the  $k$  optimal sets, there is at least one set that covers  $\geq n_t/k$  of these.

$$\text{Thus, } n_{t+1} \leq n_t - \frac{n_t}{k} = n_t \left(1 - \frac{1}{k}\right) < n_t \cdot e^{-1/k}$$

*(since  $1 - x < e^{-x}$  for  $x \neq 0$ )*

This implies that:  $n_t < n \cdot e^{-t/k}$ .

For  $t = k \ln n$ ,  $n_t < 1$  i.e.,  $n_t = 0$ .

□

*Under widely believed assumptions, no algorithm has a better guarantee!*