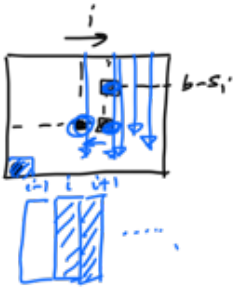
$A[b,i] = \max\left(A[b-s_i, i+1] + p_i,\ A[b, i+1]\right)$ — only if $b \geq s_i$

items $1 \sim n$ | items $i+1, \ldots, n$



$A[B, 0]$

$O(Bn)$ storage $\downarrow$ $O(B)$

```
Knapsack (b, i, ..., n) {                                   B = n.
    max { Knapsack (b - s_i, i+1, ..., n) + p_i   (only if b ≥ s_i)
        { Knapsack (b,       i+1, ..., n)
}.
```

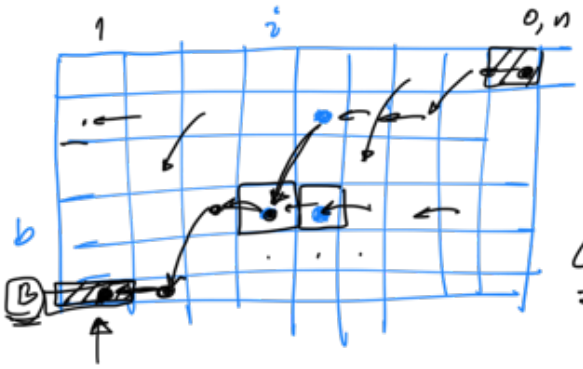
2^n ✓ n^2 ✓

bits to write down the input: x^2 x^3 ...

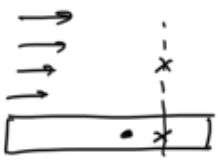
(1) Is this an efficient algorithm? NP Complete. — is this polynomial time? $O(nB)$. $n \log B$: exponential time alg.

(2) What if we also want to find out the actual set of items to pick?

$B = 10^{200} + 1$ — 201 bits.



$O(nB)$

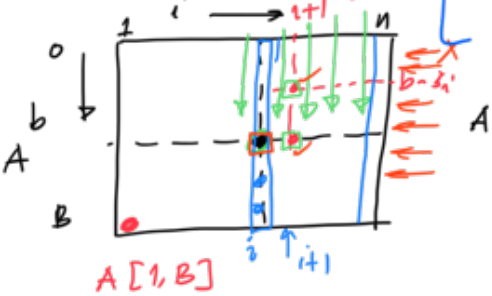


— × —

21/10/21. Knapsack

(1) Save space: to compute entries in the i^{th} column, we only need entries in the $(i+1)^{th}$ col. $O(B)$ space.

for $i = n-1$ down to 1
  for $b = 0$ to B ⊛



$A[i,b]$ = max. profit using $i, i+1, \ldots, n$ with a knapsack of capacity B.

$A[i,b] = \max\left(A[i+1, b],\ A[i+1, b-s_i] + p_i\right)$ ✓ ⊛ $O(1)$

$A[i+1, b]$: don't take i. $A[i+1, b-s_i] + p_i$: take i, $b \geq s_i$.

$B = n^2$

$\| O(Bn)$.

$1, 2, \ldots, n$

$B = 2^{\lg B}$

|| polynomial time: poly. in the

s_i, p_i B

$\sum_{i=1}^{n} (\log s_i + \log p_i) + \log B$

≥ 1 n

"size" of the input
bits needed to write down the input

(2) what if we also want to find the actual solution.
- also store the solution at each table entry: $O(n)$. $\Rightarrow O(nB)$ space.

Aside: n. $k = 2$

$(\log_2 n)$ $O(n)$.

exponential time?

```
while (k < n)
    if (k divides) output not prime
    else k++.
output prime
```

n : $\log_2 n$ 10^{200} $10^{200} + 1$

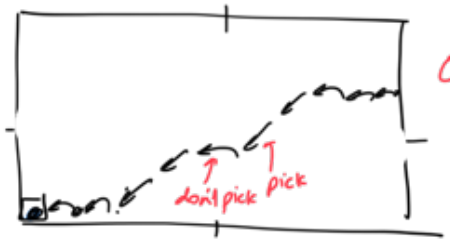
$100 \ldots 01$

200

$\times$

$A[1, B] = \max(A[2, B], A[2, B - s_1] + p_1)$.

$A[i, b]$.



$O(nB)$

(i) ← $A[i,b] = A[i+1, b]$

(ii) ↙ $A[i,b] = A[i+1, b - s_i] + p_i$

(i) $O(B)$ space if we just want the max. profit.

(ii) $O(nB)$ space if we want the actual solution.

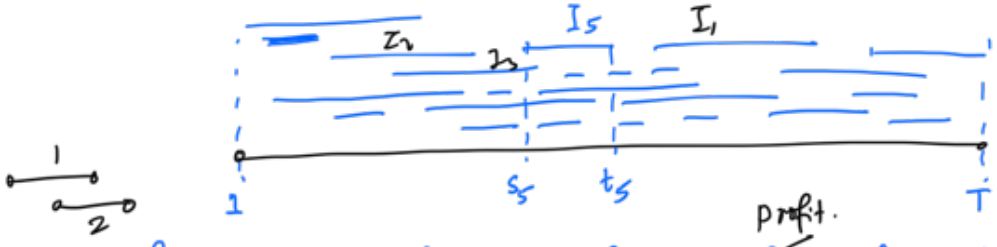
(iii) $O(nB)$ time.

Q Can we get $O(nB)$ time, $O(B)$ space and get the actual solution?

YES.

---

(2) Max. profit Interval Selection Problem:



$I_1, \ldots, I_N$

$I_i = [s_i, t_i]$.

p_i: profit of interval I_i

Pick a subset of intervals of max ~~size~~ profit. which do not overlap

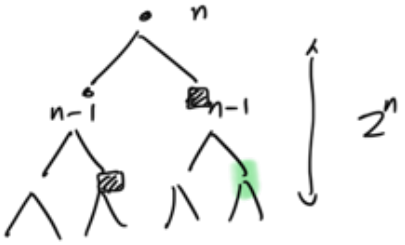
(Recursive algorithm).

Select($I_1, \ldots, I_n$) { $n=1$: output I_1. (base case).

max (select($I_2, \ldots, I_n$), p_1 + select(✓✓)).

all intervals among $I_2, \ldots, I_n$ which do not overlap with I_1

}



2^n running time.

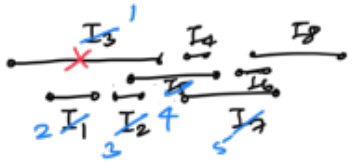
Dynamic Program: What are all possible recursive calls??

Select (any subset). ← 2^n

$I_k, \ldots, I_n$

$I_1, I_2, \ldots, I_n$

~~$t_1 \le t_2 \le t_3 \le \ldots \le t_n$~~



$s_1 \le s_2 \le s_3 \le \ldots \le s_3$

Select $(I_1, \ldots, I_n)$ {.

max (select($I_2, \ldots, I_n$), p_1 + select( ))

$I_2, I_4, I_5, \ldots, I_8$.

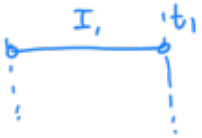
select $(I_1, \ldots, I_n)$ {

max(select $(I_2, \ldots, I_n)$, p_1 + select(intervals starting after t_1))

$I_k, \ldots, I_n$.

$s_i \ge t_1$

$I_1, I_2, \ldots, I_n$.



possible recursive calls : n.

select($I_\ell, I_{\ell+1}, \ldots, I_n$)

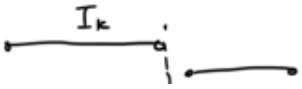
$A[n] = p_n$.

$A[k]$:= max profit when the intervals are $I_k, \ldots, I_n$

for $k = n-1$ down to 1

$A[k] = \max(A[k+1], A[\ell] + p_k)$

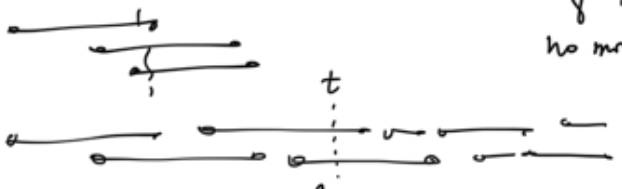
I_ℓ is the first interval starting after t_k.



$O(n \cdot \log n)$.

we have two halls instead of one hall!

Find the max. profit subset of intervals such that at any time no more than 2 such intervals contain t.

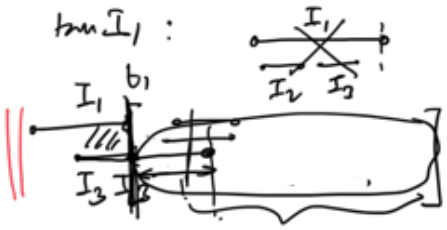


Exercise: Give a dynamic Prgr. alg. for this problem:

Select $(I_1, I_2, \ldots, I_n)$ { $s_1 \le s_2 \le \ldots \le s_n$.

don't select I_1 | select $(I_1, \ldots, I_k)$

$(=, I_i, \ldots, I_n)$ | turn I_1 :



$I_k, \ldots, I_n$

---

(3) Two companies: A, B: A ⇄ B cost C amount of money.

A: p_1^A p_2^A $\ldots$ p_n^A — p_i^A: amount of money earned if present at A on day i

B p_1^B p_2^B $\ldots$ p_n^B — p_i^B

On day 1, A. A, A, B, B, B, A, A, A, — −C, −C

Max. [total profit − Cost].

MaxProfit $(A, B, 1, 2, \ldots, n)$ {

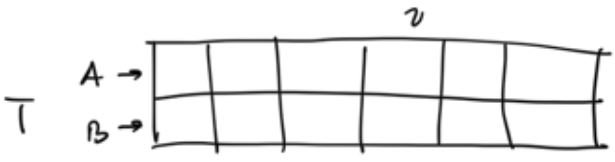
p_1^A + MaxProfit $(2, \ldots, n)$

$O(2n)$.

MaxProfit $(A, i, \ldots, n)$ {

p_i^A + max [maxprofit $(A, i+1, \ldots, n)$, maxprofit $(B, i+1, \ldots, n) - C$]

}.



= [max profit ...] if we start at A on day i

T[A, i] : max. ...
T[B, i] :

---

(4) Edit Distance Problem:

edit: (i) Add a character (ii) Delete a char.

Given two strings $s_1, \ldots, s_n$ and $t_1, \ldots, t_m$ define EDIT DISTANCE between them is defined as

the min # of edits in the first string & get to the second string.

ABABA → ABAB → A A B → ACAB.

ACAB

Another way of defining: we only insert characters in the strings (both) so that they are identical

Edit distance = # stars getting added.

A B C A B A
A [B] [C] A B [A]

A B [*] A B [A]
A [*] C A B [*]

Edit Distance (A[1...n], B[1...m]) { Base Case...

Min ((a) EditDistance(A[1... n-1], B[1...,m]) + 1, ... A[n] / .... * || (a)

OR

(b) EditDistance(A[1.. n], B[1... m-1]) + 1 .... * / ... B[m] || (b)

OR

(c) Edit(A[1...n-1], B[1... m-1]) .... A[n] / .... B[m] (c)

← only applies if A[n] = B[m]

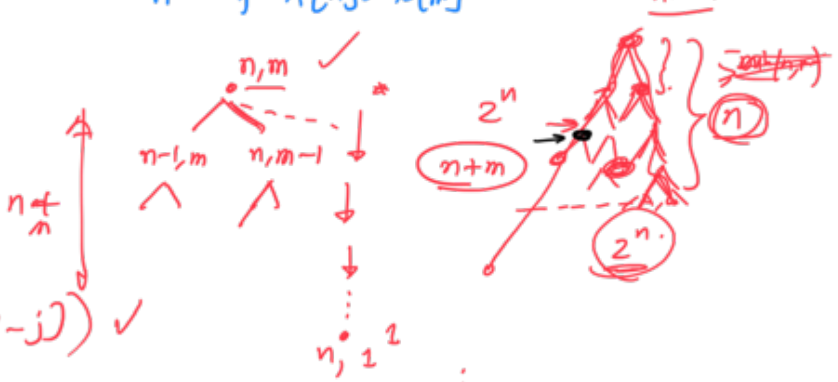
}.

exponential time.

What are the different recursive calls?

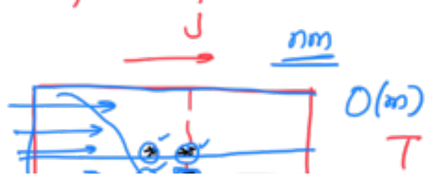
(A[1..i], B[1..j]) ✓

≤ nm ✓

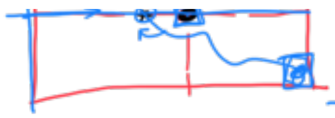


Store all of the recursive calls.

T(i, j) : store EditDistance ...



$(A[1 \ldots i], B[1 - j])$



$* \; T(i,j) = \min\big(T(i,j-1)+1, \; T(i-1,j)+1, \; T(i-1,j-1) \big)$ ← only if $A[i] = B[j]$.

$i=0, j=0$ $\quad T(0,j) = j, \; T(i,0) = i$.

for $i = 1, \ldots, n$
 for $j = 1, \ldots, m$
 (*)

$O(nm)$ time.

$A[1..n]$ $\quad A[1 - i]$
$B[1..m]$ $\quad B[1 - j]$

i, j, k

Maximum Flows:

electrical current, water, traffic....

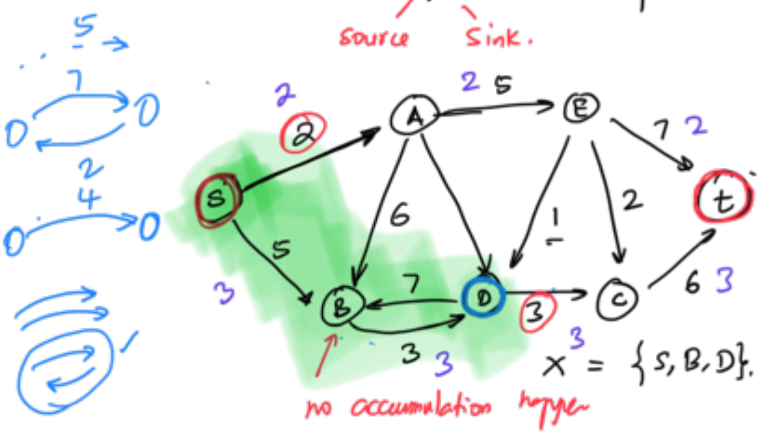
directed graph

Problem : G, e has a capacity $u_e \geq 0$

s, t : two special vertices
(source, sink.)

- Greedy
- Divide & Conquer
- Dynamic Programming
- Maximum Flows.

Convention / Notation: No edge enters s, No edge leaves t.



- Electric current:
  each edge is a wire
  u_e : max. current that can flow one.
- Water pipes:
  each edge is a water pipe
  u_e: rate at which water can flow on this pipe.
- traffic : each edge is a road.
  u_e: how much traffic flows on each edge (rate of traffic)
  eg, 10 trucks / hour.

What do we want?

Defn: A flow f specifies a quantity f_e for each edge :

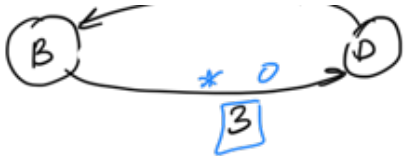
(i) $0 \leq f_e \leq u_e \quad \forall$ edge e.

(ii) "flow conservation"
for every vertex v other than s or t,

$$\sum_{e:\, e \text{ comes into } v} f_e = \sum_{e:\, e \text{ goes out of } v} f_e$$

$\delta^-(v)$: edges entering v.
$\delta^+(v)$: edges leaving v.

7 & 4



??

??

Let f be flow.

flow going out of s := $\sum_{e \text{ leaving } s} f_e$

flow coming into t = $\sum_{e \text{ entering } t} f_e$

value of the flow ≡ flow going out of s.

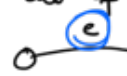
Claim: Flow coming into t = Flow going out of s.

Pf: For every vertex $v \neq s,t$

$$\sum_{e \in \delta^-(v)} f_e = \sum_{e \in \delta^+(v)} f_e \quad \forall v$$

total flow entering v = total flow leaving v

Add all of these equations:



e appears on the RHS for v and on the LHS for w.

$$\sum_{e' \in \delta^-(w)} f_{e'} = \sum_{e' \in \delta^+(w)} f_{e'} \quad : w$$

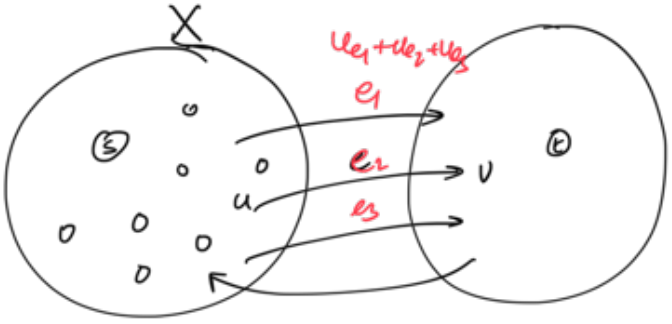
appears on the RHS for w.

When we add all of the equations,

any such edge appears on the LHS for v

$$\sum_{e \in \delta^+(s)} f_e = \sum_{e \in \delta^-(t)} f_e$$

Goal: To find a flow of maximum value



Defn: A s-t Cut is a set of vertices X such that X contains s and does not contain t.

We say that $e = (u,v)$ leaves X if $u \in X$, $v \notin X$.

$e = (u,v)$ enters X if $u \notin X$, $v \in X$.

$$\text{Capacity}(X) = \sum_{e \text{ leaving } X} u_e$$

Claim: max. flow from s to t = capacity(X). ∎

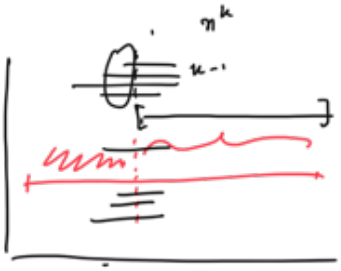
Pf: For every vertex $v \in X$, $v \neq s$

$$\sum_{e \in \delta^-(v)} f_e = \sum_{e \in \delta^+(v)} f_e.$$

Add all of the conditions

$$\sum_{e \in \delta^+(s)} f_e = \underbrace{\sum_{\substack{e \text{ going out} \\ \text{of } X}} f_e - \sum_{\substack{e \text{ coming into} \\ X}} f_e}_{\geq 0} \leq \sum_{e \text{ going out of } X} u_e = \text{capacity}(X).$$

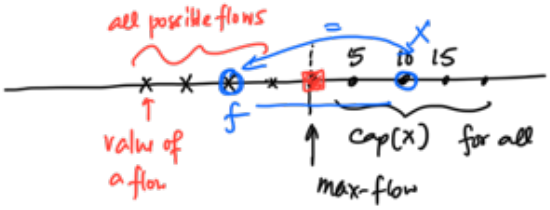
$\forall X$: $\min_X$ capacity(X)



28/10/21

75% post mins 1.

Every cut X gives us an upper bound on the max flow from s-t.



Suppose we find a flow f and an s-t cut X s.t. value of f = cap(X). ⇒ f is max-flow & X is min cut.

max flow $\leq$ $\min_X$ cap(X). : is there equality here?

$=$ X: X is an s-t cut

Thm: max flow from s-t = min capacity of a cut. ∎ ✓

Alg: start by building flow incrementally, and stop when its value = capacity of a cut.

(i) Start with $f_e = 0$ on all edges. ✓

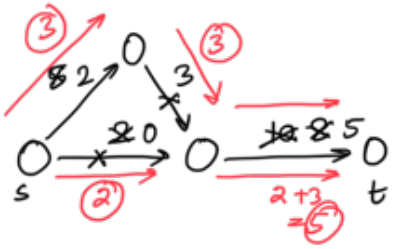
(ii)

continue
- Find a path from s to t.
- P: Let c be the min. capacity of an

edge in P. Increase/Send c amount of flow on all edges in P.

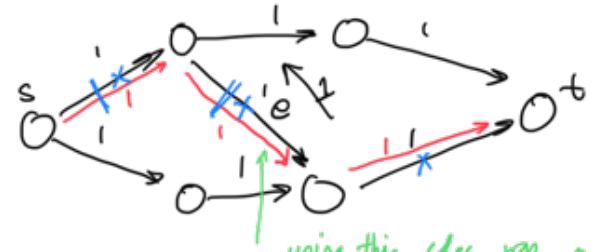
- update the capacity of every edge to $u_e - f_e$: remove edges of capacity 0.

residual capacity.



Is this going to find the maximum flow? ND

(11)



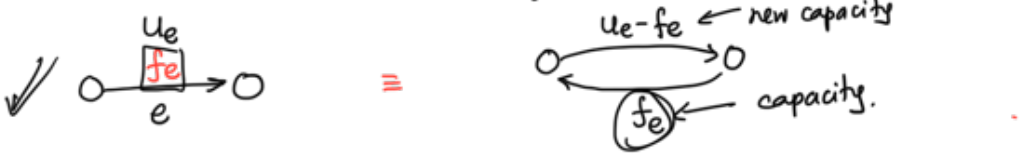
max flow = 2.

using this edge was a "mistake"

Modify the algorithm:

New Idea:

Allow the algorithm to "undo" a mistake.



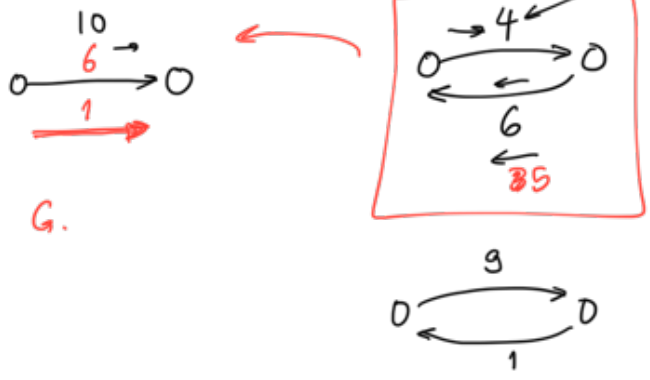
G_f

G, flow f

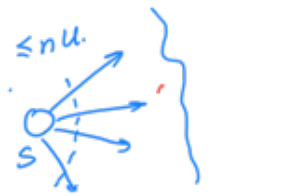
Residual graph. ✓

residual cap.

Ex:



G.



G, fe

Initialize $f = 0$, $G_f = G$

repeat:

O(m+n) time

find a path P from s to t in G_f.

Let c be the smallest residual capacity of an edge in P.

Send c amount of flow along P.

update f in G, update G_f.

(i) Construct G_f $O(m+n)$

(ii) s − t in G_f $O(m+n)$

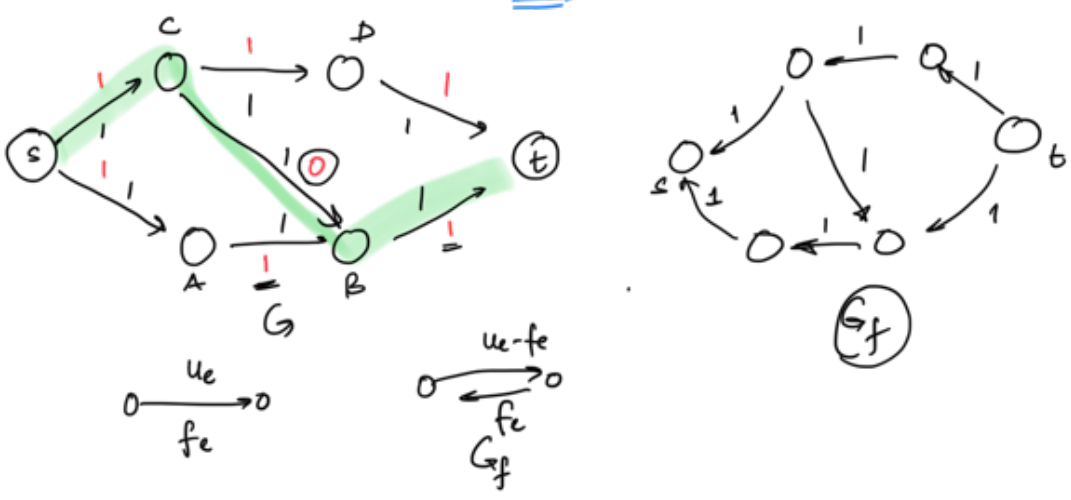
(iii) $O(n)$

flow inc. $0 \to nU$

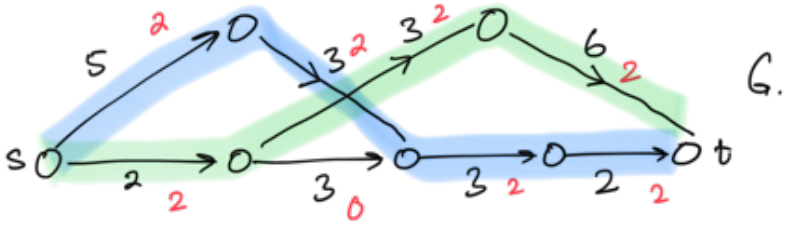
≥ 1 unit

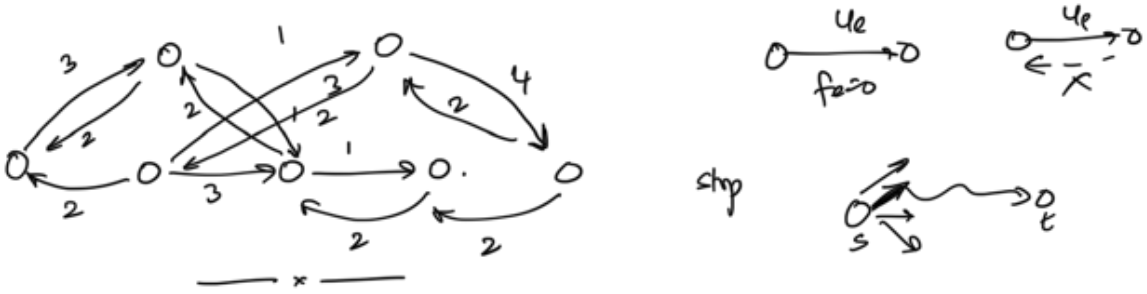
$\Rightarrow \leq nU$ iterations. ✓

$O(n^2)$.



Another Example:





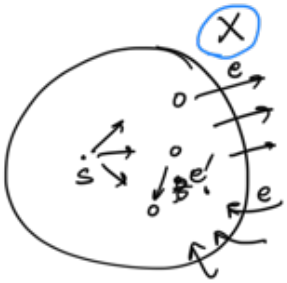
whenever we find a path from s to t in G_f, we increase the flow from s to t

the process stops when there is no path from s to t in G_f.

Claim: If there is no path from s to t in G_f, then f is a max. flow.

Pf: (Last time we had shown)

If f is any flow, $\forall$ x, f $\quad$ value(f) = cap(x)

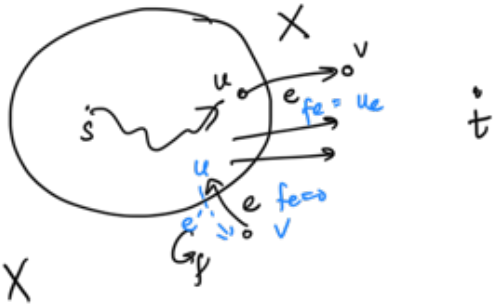


flow out of X = flow going out of X − flow coming into X

$$\text{value}(f) = \sum_{e \text{ leaving } X} f_e - \sum_{e \text{ entering } X} f_e \overset{\text{a}}{\checkmark} = \sum_{e \text{ leaving } X} u_e = \text{cap}(X).$$

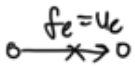
Suppose there is no path from s to t in G_f.

Let $X = \{v : \text{there is a path from } s \text{ to } v \text{ in } G_f\}$.



$s \in X,\ t \notin X$

(i) Let e be an edge leaving X : why isn't v also in X?
e cannot be present in G_f. $f_e = u_e$



(ii) e enters X : e' be the reversal of e. e' is not present in G_f (else $v \in X$)

$f_e = 0$

f is a max flow & X is a min-cut. max flow $\le nU$.

Ford-Fulkerson Alg: Assume all capacities are integers, U: max. cap. of an edge.

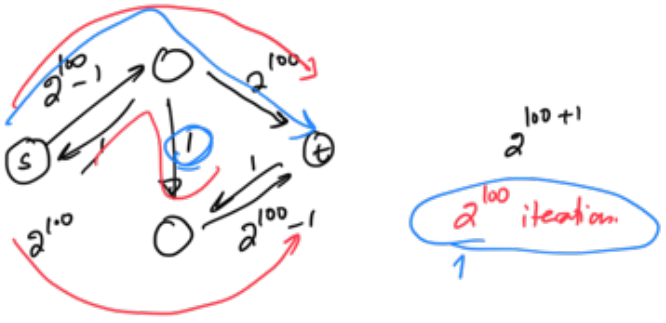
In each iteration: - form G_f $O(m)$ time.
- increase the flow by ≥ 1 unit

$\le nU$ iterations, $O(mnU)$ time.

$O(mnU)$ time. 2^{100}

Comment: all cap. are integer. $1, U$.

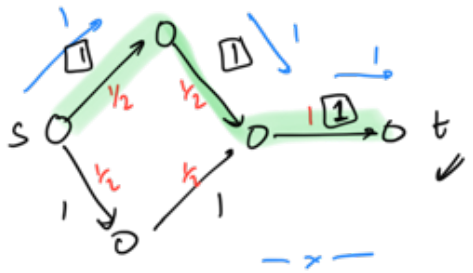
(i) if U is large, then alg. can take lot of time.



- Shortest path in G_f (use BFS) : $O(mn^2)$ time.
- find a path where the min. residual cap. is as large as

possible. $O(mn^2 \log U)$ time.

(ii) If all capacities are integers, then the alg. always deals with integers.
⇒ the max flow found by the alg. sends integral amount of flow on each edge.

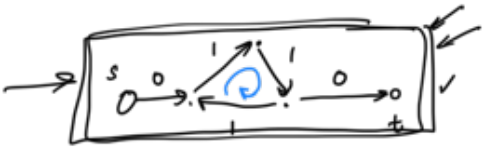


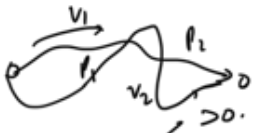
↓
there is a max-flow which only sends integer flows.

Integrality of Max-Flow.

—×—

1/11/21. (i) Max flow = Min. cut, alg. for finding these.
(ii) Integrality of max-flow. : there is a max-flow which is integral.
(iii) How do we think of a flow?? Can we think of a flow as consisting of flows along path?

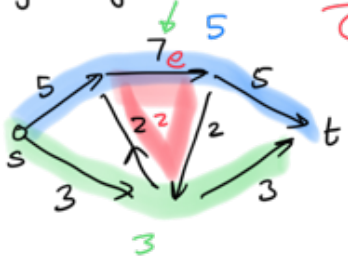


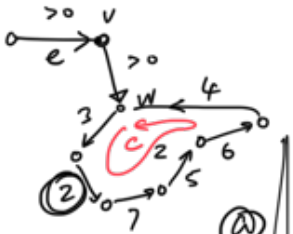


Thm [Path decomposition of flow] Let f be any flow in G. Then we can find s-t paths $P_1, \dots, P_\ell$ and values $v_1, \dots, v_\ell$ and cycles $C_1, \dots, C_k$ and values $w_1, \dots, w_k$ such that for every edge e,

$$f_e = \sum_{P_i \text{ contains } e} v_i + \sum_{C_j \text{ contains } e} w_j \quad : k+\ell \le m.$$

Pf: H: subgraph of G consisting of edges with $f_e > 0$.



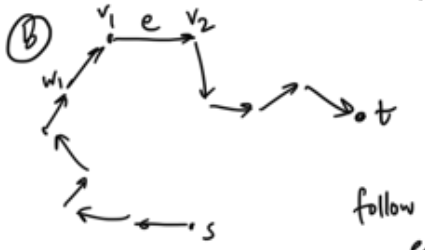


$e \in H$

(i) Repeat a vertex : we have a cycle C with $f_e > 0$ on all edges.

(ii) Min. f_e on an edge in C and subtract Δ from each edge in C.

≥1 edge drops out of H

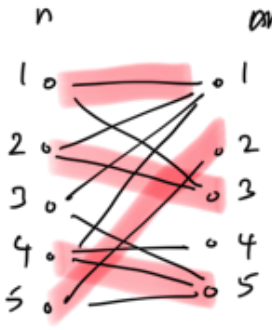


follow how in coming
edge from s either find a cycle in H or a walk from s-t

Applications of Max flow:

$U = 1$ $O(mn)$.

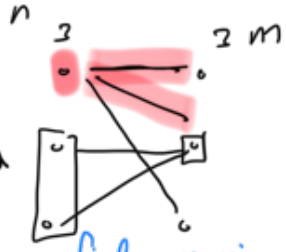
① Bipartite Matching:



Matching: subset of edges which do not share a common vertex.

e.g. V_1: set of student
V_2: set of hotel rooms.

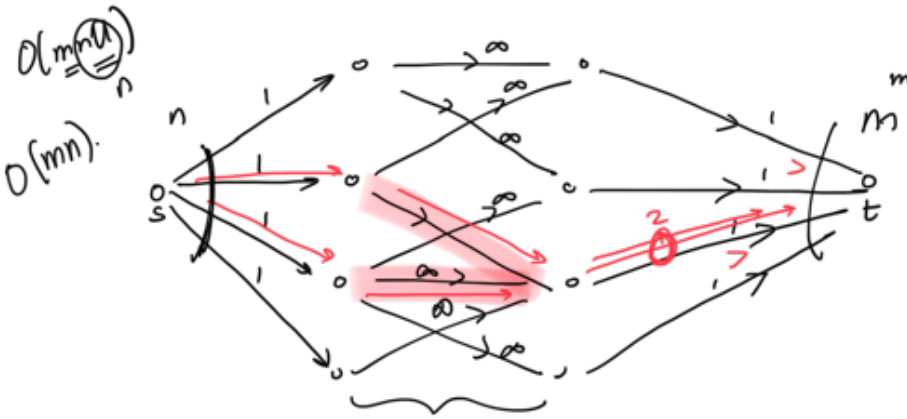
- Can we have a matching of size n ??
- What is the max. size of a matching?



2.

s, t are directed

How do we use max flow to find maximum matching?



P → P'
matching → max-flow.

$O(m\sqrt{n})$

Thm: there is a flow of value k from s to t in G' if and only if there is a matching of size k in G.

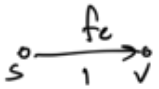
Pf: (⇐) Suppose there a matching of size k in G. : for every edge $e = (u,v)$ in the matching, send 1 unit of flow along

Pe: s, u, v, t

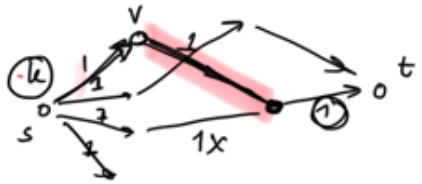
All of them paths are disjoint.

(⇒) Suppose there a flow f of value k from s to t.

we can assume that f is integral



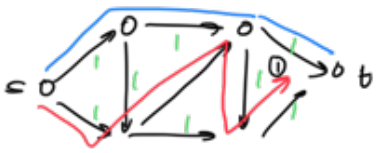
$f_e = 0$ or 1.



Look at the edges between V_1 and V_2 which are carrying 1 unit of flow.

There will be k such edges and they will form a matching.

② Disjoint paths: given a directed graph G. We say that two paths



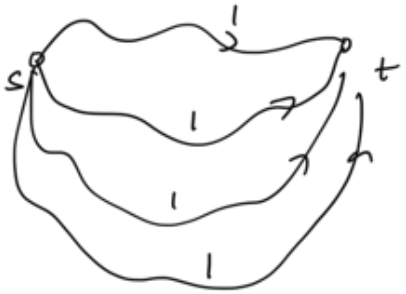
from s to t are "edge" disjoint if they do not share a common edge.

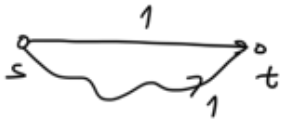
- we want to find the maximum no. of edge disjoint paths from s to b.

Place a capacity of 1 on every edge find max-flow from s to b.

Claim: there is a flow of value k from s to t if and only if there are k edge disjoint path from s to b.

Pf: ($\Leftarrow$) Suppose are k- edge- disjoint paths from s to t.





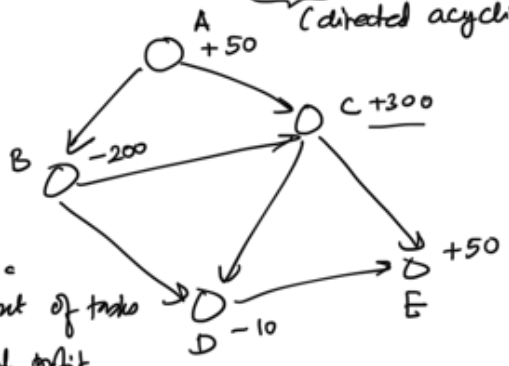
($\Rightarrow$) Suppose there is a flow of value k from s to b. (follows from path decomposition).

(3) Application of min-cut:

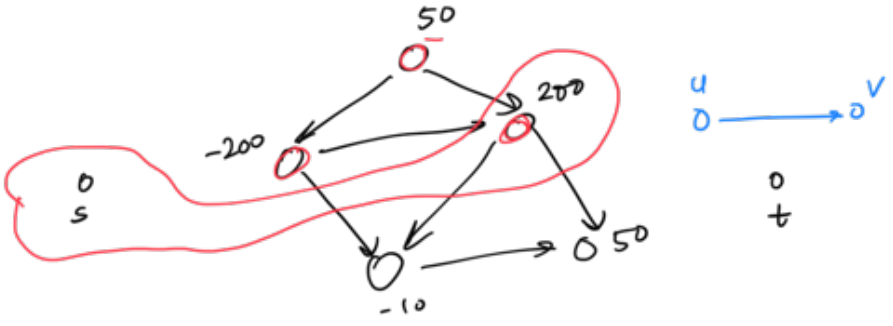
Project Selection Problem: DAG. precedence constrain tasks (directed acyclic graph).

Every task either generates or costs money.

v : p_v



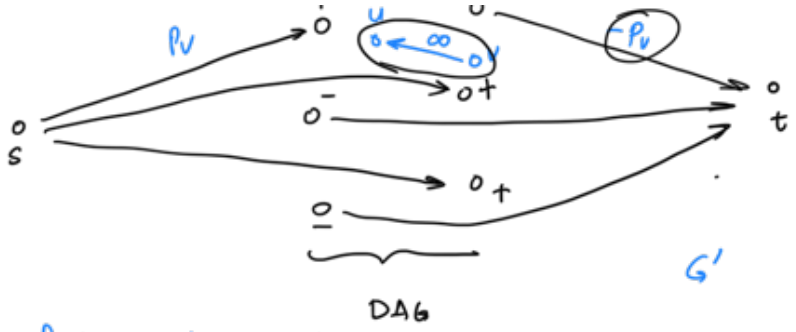
Q: we want to perform a subset of tasks such that the total profit is maximized.



Idea: Add a vertex s, t ---- min-cut :

How do we ensure that the min-cut is valid ??

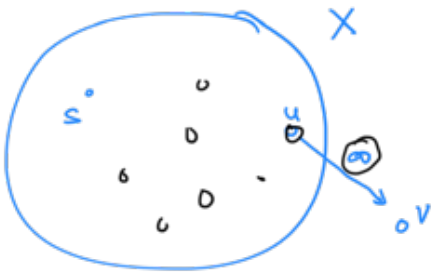
if $(u,v) \in G$



then in G', we add (v,u) with ∞ capacity.

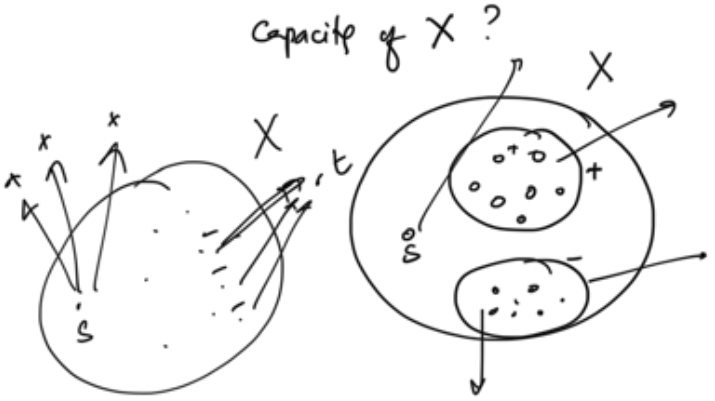
DAG

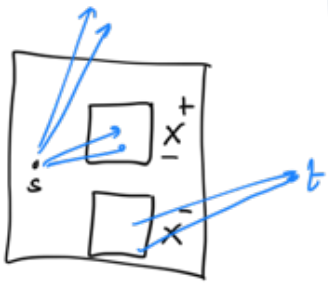
Claim: find a min-cut for separating s and t.



Let X be the min-cut.
$u \in X$. $v \to u$ in G.
if $v \notin X$, then (u,v) has cap. ∞. The cut X has ∞ capacity.

Capacity of X ?





V^+: vertices with +ve profit
V^-: vertices with −ve profit

$$\infty \quad \underbrace{cap(X)}_{a} = \sum_{v \in V^+ - X^+} p_v + \sum_{v \in X^-} (-p_v) \qquad -(-5\infty) = 5\infty \ \checkmark$$

$$= \sum_{v \in V^+} p_v - \sum_{v \in X^+} p_v - \sum_{v \in X^-} p_v$$

$$= \underbrace{\sum_{v \in V^+} p_v}_{\text{fixed number.}} - \underbrace{\left[\sum_{v \in X} p_v \right]}_{\text{net profit of } X.} \checkmark$$

Min of $cap(X)$ = max profit of X.

$a + b = C$.

$$= \underbrace{\sum_{v \in V^+} p_v}_{=C} - \underbrace{\sum_{v \in X} p_v}_{\text{net profit of } X.} \qquad a + b = C$$