Jiankun Dong.

Finals:

① Problem 1:

Preprocessing:

① create an array1 [0, 1, 2 ..., k], where total length is k, each element in array is set to 0.

② go through the n integers, for each integer x, if it's in the array from step 1, increase array1 [x] by 1

③ then update array1, for j from the second element to the last (2nd k), array1 [j] = array1 [j] + array1 [j-1], so that array1 [j] is the sum of all previous elements.

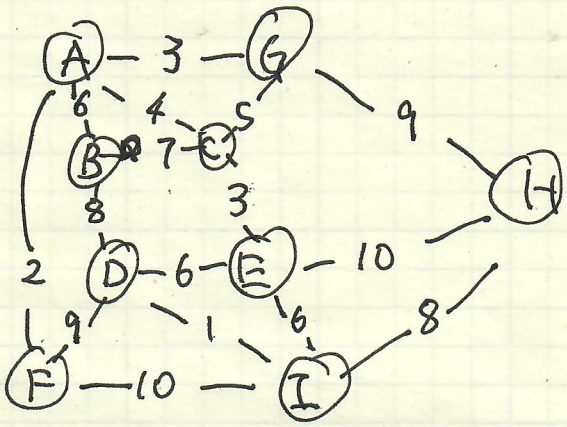
Execution:

for range [a, b], return array1 [b] - array1 [a-1]

O(1).

2) Start with A for Prim's

a) KRUSCAL'S



9 edges

Preprocessing/sorting:

{DI:1, AF:2, AG:3, CE:3, AC:4, CG:5, AB:6, DE:6, EI:6,

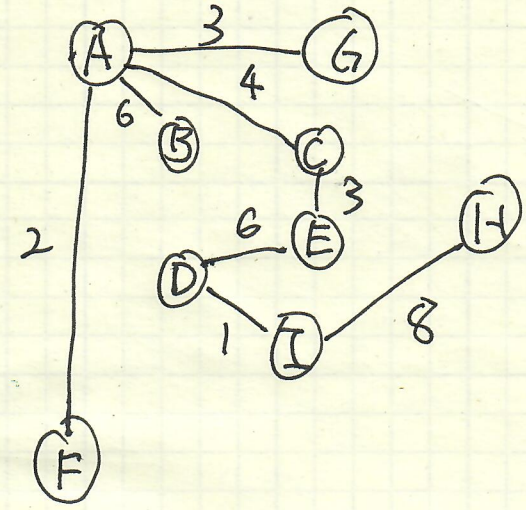
BC:7, BD:8, IH:8, BDF:9, GH:9, EH:10, FI:10}

- Step 1: DI
  {DI}, no loop
- Step 2: AF
  {DI, AF}, no loop
- Step 3: AG
  {DI, AF, AG}
  no loop
- step 4: CE
  {DI, AF, AG, CE}
  no loop
- step 5: AC
  {DI, AF, AG, CE, AC}
  no loop
- step 6: CG
  LOOP, skip
- step 7: AB
  {DI, AF, AG, CE, AC, AB}
  ~~AD. loop~~ No loop
- step 8: DE
  {DI, AF, AG, CE, AC, AB, DE}
  no loop
- step 9: EI
  loop, skip
- step 10: BC
  loop, skip
- step 11: BD
  loop, skip
- step 12: IH
  {DI, AF, AG, CE, AC, AB, DE, IH}
  no loop.
  added edge count = 8 = 9-1
  <<terminate>>

* result see next page.

Total weight = 1+2+3+4+6+6+8
= (33).

tree:



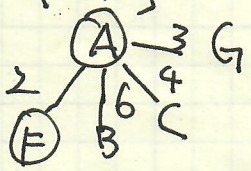
total weight = (33)

KRUSCAL'S.

2)b) Prim's.

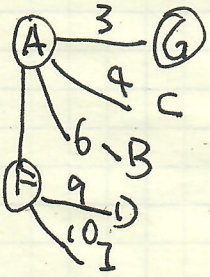
- Step 1: add F

{AF}



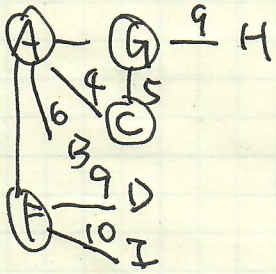
- step 2: add G

{AF, AG}



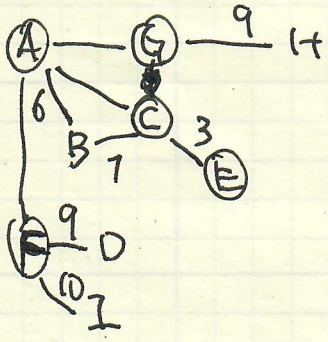
- step 3: add C

{AF, AG, AC}



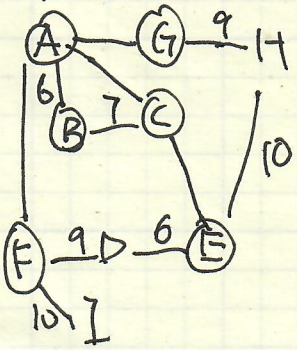
- step 4: add E

{AF, AG, AC, CE}

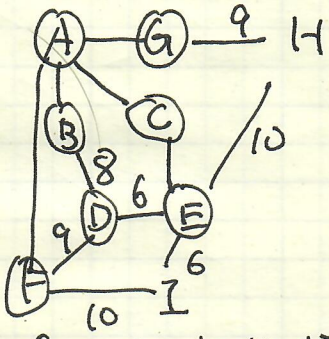


- step 5: add B

{AF, AG, AC, CE, AB}

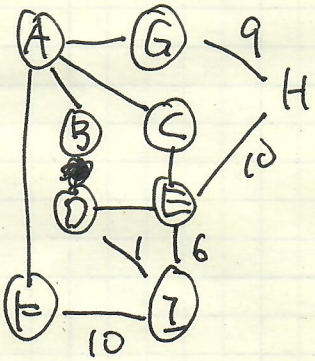


- step 6: add D



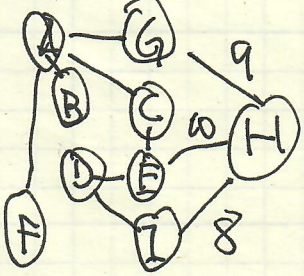
{AF, AG, AC, CE, AB, DE}

- step 7: add I



{AF, AG, AC, CE, AB, DE, DI}

- step 8: add H

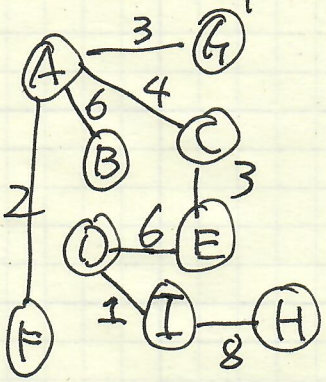


{AF, AG, AC, CE, AB, DE, DI, IH}

9 vertices

<<terminate>>

- Final Graph:



- total weight:

1+2+3+3+4+6+6+8

= (33).

(5) Dijkstra's

Step 1:
path {AC}
unvisited {B, D, E}

Distance: A: 0
B: AB=4
C: AC=2
D: ∞
E: ∞

Step 2:
path {AC, AB}
unvisited {D, E}

Distance: A: 0
B: AB=4, ACB=5
C: AC=2
D: ACD=7
E: ACE=6

Step 3:
path: {AC, AB, CE}
unvisited: {D}

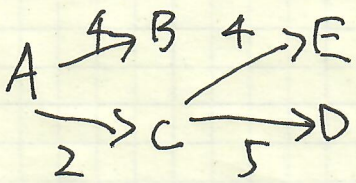
Distance: A: 0
B: AB=4
C: AC=2
D: ACD=7, ABD=7
E: ACE=6, ABE=6

Step 4:
path: {AC, AB, CE, CD}
unvisited: {}

Distance: A: 0
B: AB=4
C: AC=2
D: ACD=7, ABD=7,
E: ACE=6

Distance: A: 0
B: 4 (AB)
C: 2 (AC)
D: 7 (ACD)
E: 6 (ACE)

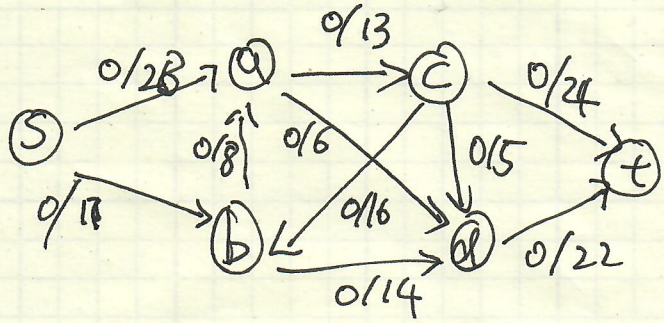
Final Graph:



(6) Ford-Fulkerson.

Answer: (33) max flow.

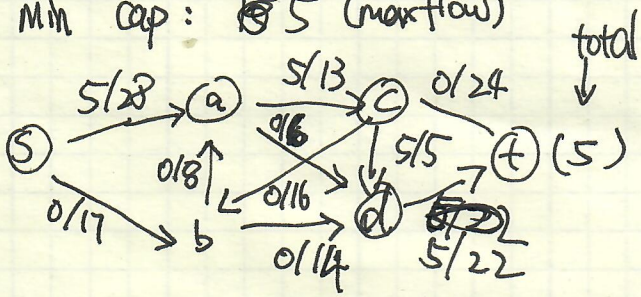
Initialization:



Step1:

path s a c d t.

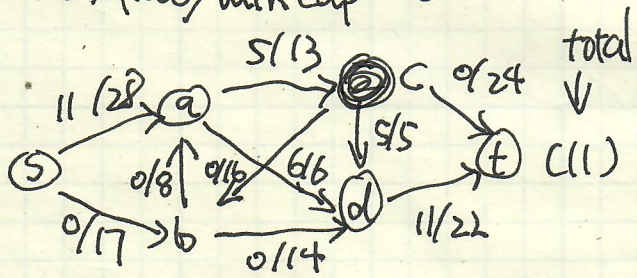
Min cap: ~~6~~ 5 (maxflow)



Step 2:

path s a d t

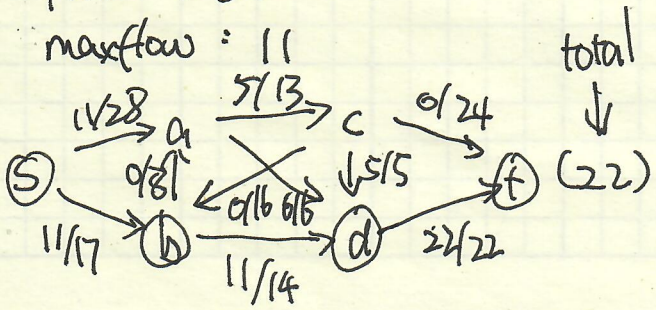
maxflow/min cap: 6



Step 3:

path s b d t

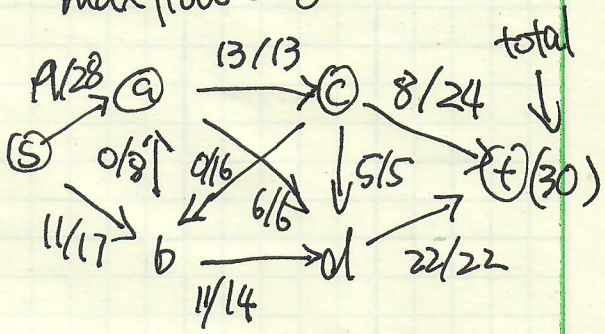
maxflow: 11



Step 4:

path s a c t

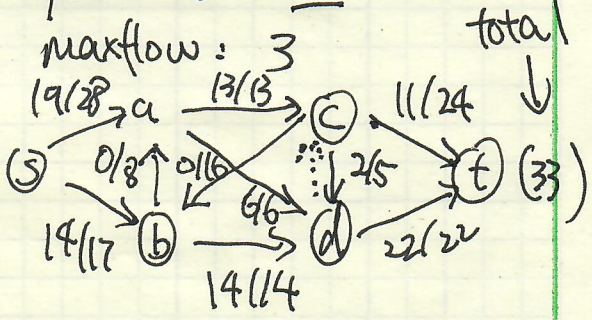
maxflow: 8



Step 5:

path s b d c t

maxflow: 3



Can't get to t after this

<< terminated >>

Max value: s->t [33]

- no available non-empty backward path from s->t
- no available non-full forward path from s->t

thus according to the algorithm, [33] is max flow value.