A network of n servers under your supervision is modeled as an undirected graph G=(V,E) where a vertex in the graph corresponds to a server in the network and an edge models a link between the two servers corresponding to its incident vertices. Assume G is connected. Each edge is labeled with a positive integer that represents the cost of maintaining the link it models. Further, there is one server (call its corresponding vertex as *S*) that is not reliable and likely to fail. Due to a budget cut, you decide to remove a subset of the links while still ensuring connectivity. That is, you decide to remove a subset of E so that the remaining graph is a spanning tree. Further, to ensure that the failure of *S* does not affect the rest of the network, you also require that *S* is connected to exactly one other vertex in the remaining graph. Design an algorithm that given G and the edge costs efficiently decides if it is possible to remove a subset of E, such that the remaining graph is a spanning tree where *S* is connected to exactly one other vertex and (if possible) finds a solution that minimizes the sum of maintenance costs of the remaining edges.

You are given a flow network with source s, sink t and edge capacities. Further, for every vertex v in the network, you are given a non-negative number d_v which is the vertex capacity. Design a polynomial time algorithm that finds a flow of maximum value that (in addition to satisfying the usual edge capacity constraints and flow conservation constraints) satisfies the constraint that for every vertex v, the flow into v is at most d_v.

Prove that the problem of deciding if a given graph G contains a simple cycle that visits at least half of the vertices in G is NP-complete.

Alice and Bob worked in a restaurant and received n currency notes in total as tips. Every note has a value (either $1, $5 or $10) written on it. The currency notes are arranged from left to right on a table in a fixed but arbitrary sequence. In particular, they are not necessarily sorted according to value. Alice and Bob play the following game to split the tip money. Alice and Bob take turns to play and at each turn, the player chooses either the leftmost currency note or the rightmost currency note and takes it. Bob is greedy and always plays using the following strategy; “If the rightmost note has value larger than the leftmost, then take the rightmost. Otherwise take the leftmost". Design an efficient algorithm that determines the plays for Alice such that the tip money Alice gets is maximized. Assume n is even and Alice plays the first turn.

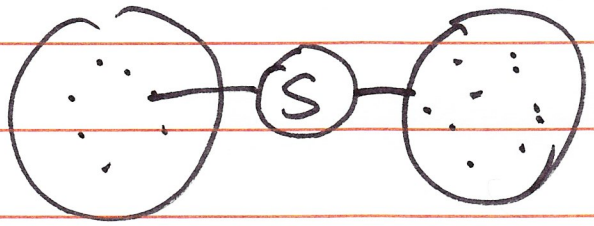
P-1. Graph Problem

To reduce edges

Two requirements.

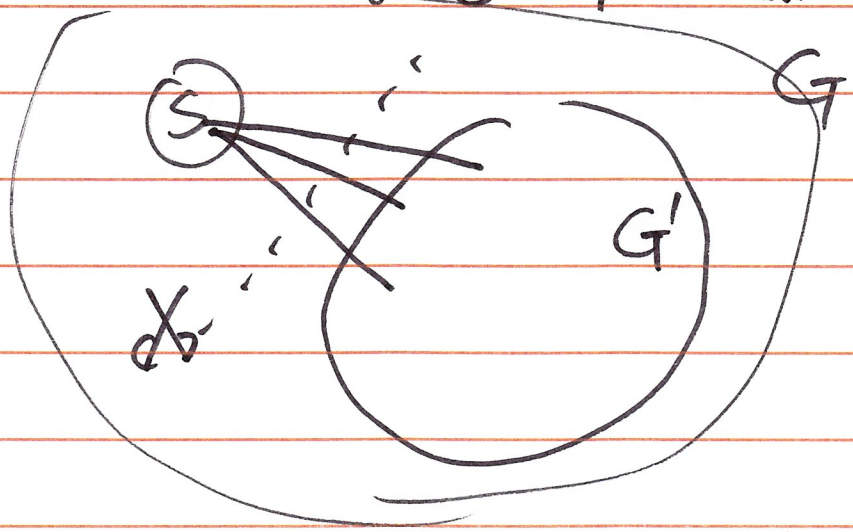
1. conectivity + minimun cost.
2. S has only ONE connection

Cannot run MST directly:



Steps:

1. disconnect S w/ remaining G ⇒ G'



2. Check connectivity of G'
3. run MST on G', eg. Prime's
4. reconnect S w/ G', choose min-cost edge.

MST alg (10 pt).

S if a leaf (5 pt)

connectivity (5 pt).

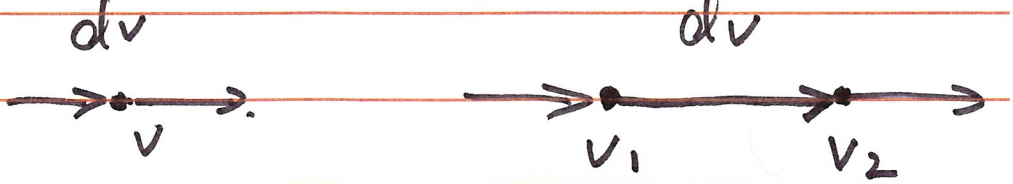
P-2. Network.

Max flow w/ vertex capacity.

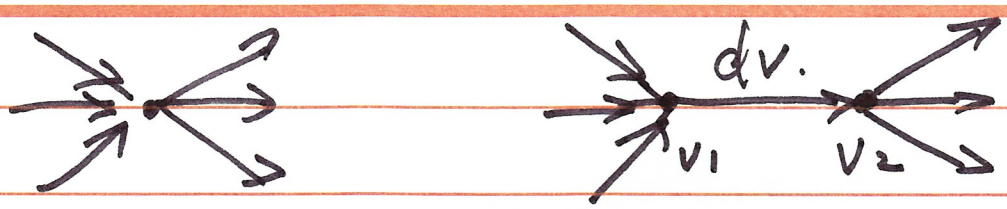
New: Vertex capacity. ?

Learned: edge capacity

⇒ convert vertex cap. to edge cap.



⇓.



⇒ Process all vertex w/ v. cap.

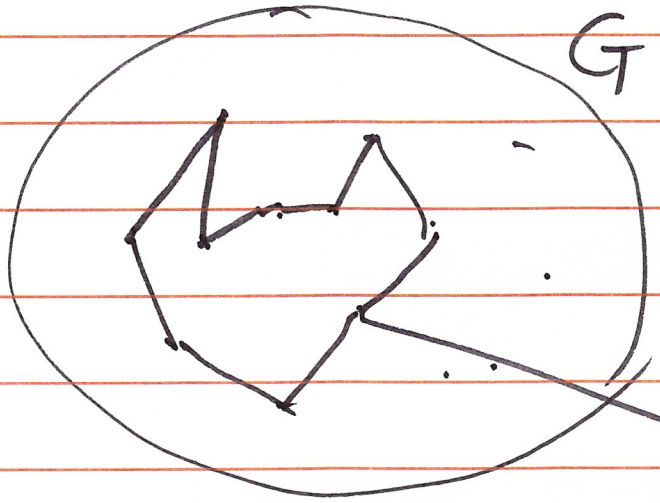
⇒ New network.

⇒ Run Ford-Fulkerson (polynomial time).

P-3. NP-completeness.

simple cycle (SC) problem

→ NPC.



of ~~edges~~ vertex.

$\geq |V|/2$

① certificate:

(1) all vertex, all edges $\in G$

(2) Form ONE Simple cycle.

(BFS/DFS)

(3) # of vertex $\geq |V|/2$

2. NP-C

Common step: reduce the problem to a known NP-C problem.

Similar to Ham-Cycle (HC)
(NP-C)

Ham-Cycle: travel each vertex in one graph exactly once.

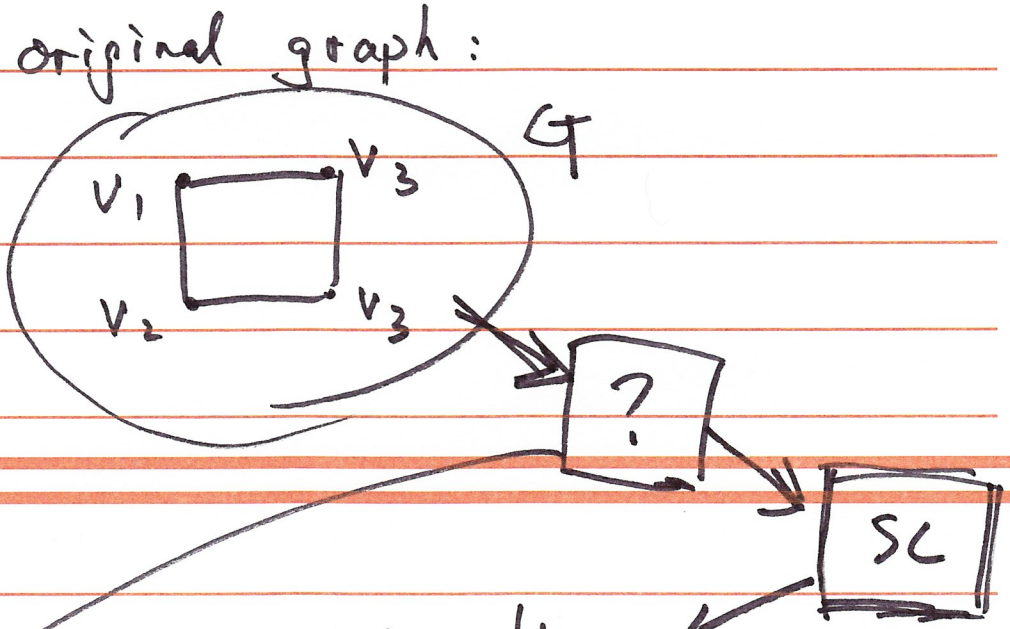
To show: SC is equavalent or harder than HC problem.

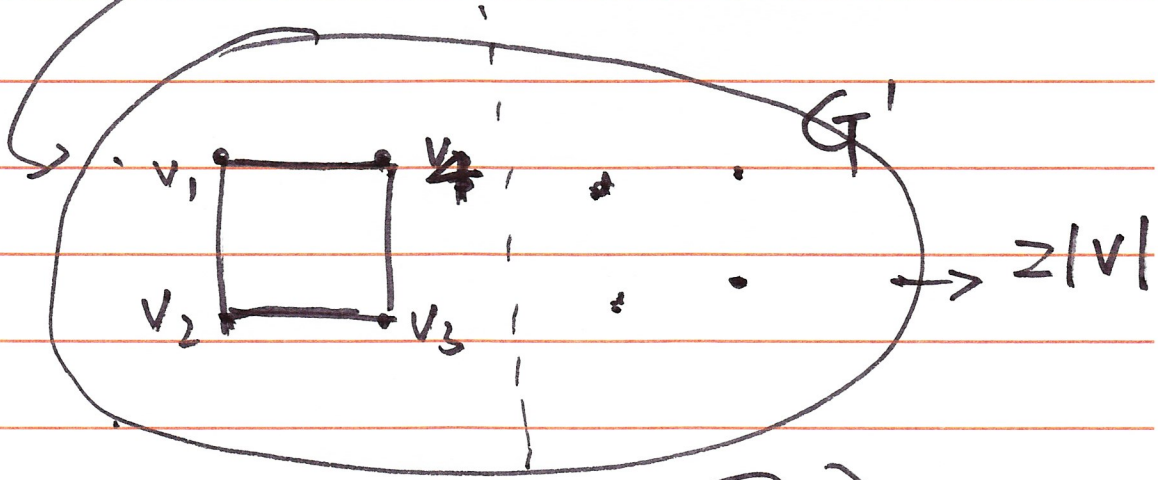
can be reduced.

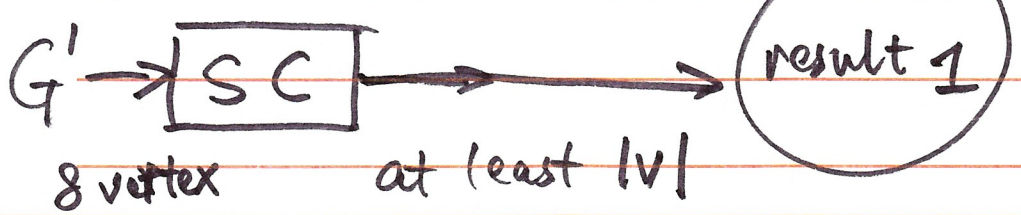
requirements

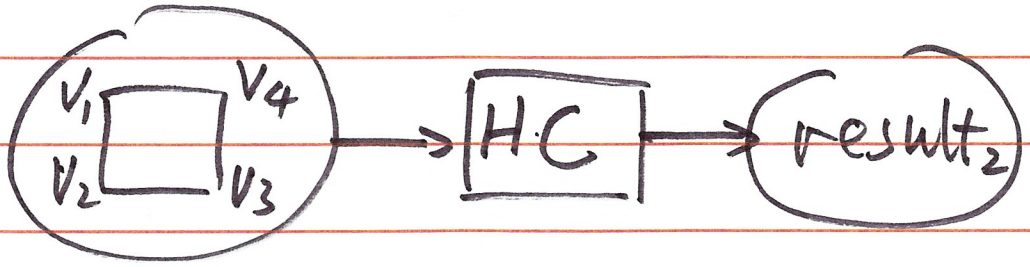
SC: $\geq |V|/2$

? → HC: $= |V|$









result$_1$ = result$_2$.

$\Rightarrow$ reduce SC to HC.

(NP-C) (NP-C).

grading:

certificate 3 pt.

NP-C 12 pt.

P-4. Dynamic Programming.

Alice

Bob

Bob's strategy: max (Leftmost, rightmost).

if leftmost == rightmost.

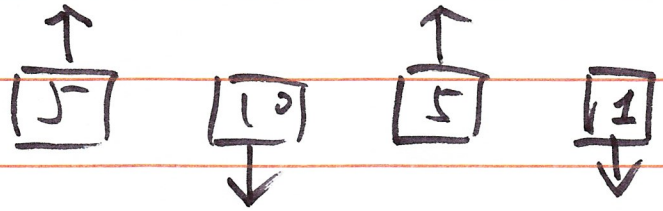
take left most.

⇒ maxmize Alice's money.

Greedy ⇐ common mistake

G1: pick biggne of left/right- most money

Alice

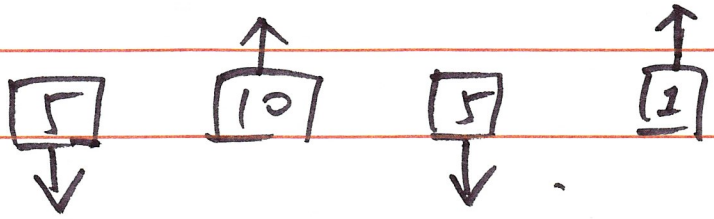


$sum_a = \$10$

Bob

$sum_b = \$10$.

Alice



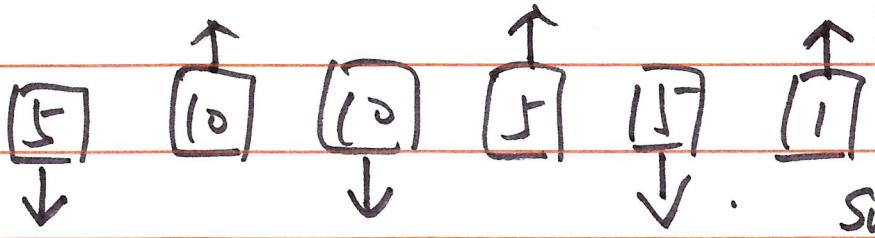
Sum a = $11

Bob

Sum b = $10

Greedy 2: minimize value Bob can get.

Alice

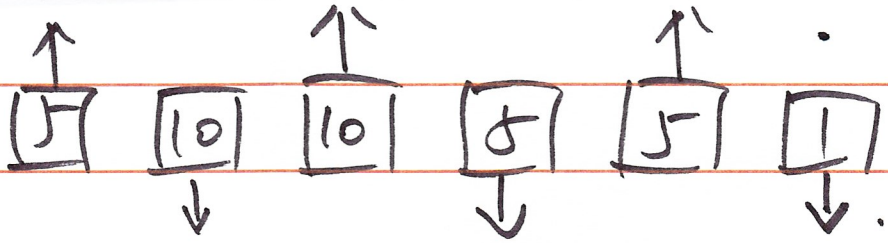


Sum a = $16

Bob

Sum b = 20

Alice



Sum a = $20

Bob

Sum = $16.

Tips: n is even #.

t_1 t_2 .... t_n

One instance after a few rounds.

$t_i, t_{i+1}, t_{i+2}, \ldots, t_{j-1}, t_j$

$i \in [1, n-1]$, $j \in [2, n]$, $i < j$

$$OPT(i,j) = \max\left[TakeLeft(i,j), TakeRight(i,j) \right]$$

$$TakeLeft(i,j) = t_i + \begin{cases} OPT(i+2,j), & \text{if } t_{i+1} \geq t_j. \\ OPT(i+1,j-1), & \text{if } t_{i+1} < t_j \end{cases}$$

$$TakeRight(i,j) = t_j + \begin{cases} OPT(i+1,j), & \text{if } t_i \geq t_{j-1} \\ OPT(i,j-2), & \text{if } t_i < t_{j-1} \end{cases}$$

Basic Case. $OPT(t_i, t_{i+1})$

$= \max(t_i, t_{i+1})$.

n is even
K is odd

Algorithm: $K = j - i$

For $K = 1, 3, \ldots n-1$

  For $i = 1, 2, \ldots n-K$

    compute $OPT(i, i+k)$;

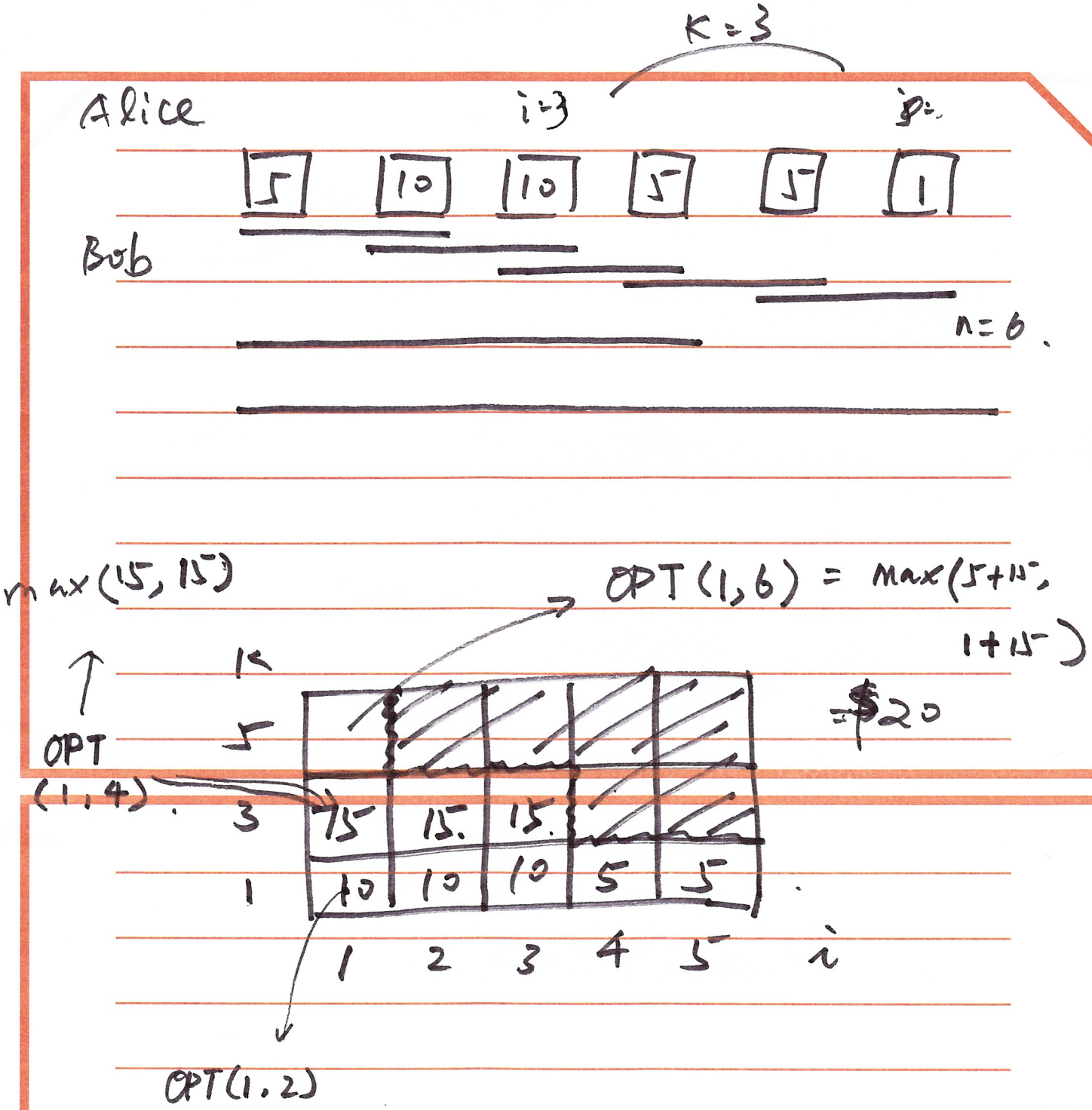
  End

End.

$\Rightarrow$ Final solution : $OPT(1, n)$

Fill a table to solve it.

Define OPT	3pt.
OPT eqn	8pt.
Base case	3pt.
Algorithm	6pt.


K=3
Alice
i=3
5
10
10
5
5
1
Bob
n=6.
max(15, 15)
OPT(1,6) = max(5+15, 1+15)
=$20
OPT
(1,4)
15
5
3
1
15
15
15
10
10
10
5
5
1
2
3
4
5
i
OPT(1,2)