MINOR-I

COL 351: Analysis and Design of Algorithms
Minor Exam

Timing: 9:10 am - 10:40 am

Important Guidelines:

1. This is a closed book exam. You cannot look at your notes or browse the internet.
2. Each solution must start on a new page.
3. You can directly reference an algorithm / theorem proved in the lectures.
4. Total marks are 42.
5. The scanned solution **must be** uploaded on Gradescope by 10:55 am sharp. The late submission deadline is 11:00 am, No submissions will be accepted later in any circumstance.

1 Short Questions [$3 \times 5 = 15$ marks]

(a) Prove that edit-distance between two strings $A,\ B$ of size respectively m, n is always bounded by $(m + n - 2|LCS(A, B)|)$, where $|LCS(A, B)|$ is the length of longest common subsequence of A and B (when these strings are viewed as sequences).

Remove those characters in A that aren't in $LCS(A, B)$ and add back the characters present in $B - LCS(A, B)$ at apt positions. Total number of edits are $(m + n - 2|LCS(A, B)|)$.

(b) An edge (x, y) in an undirected graph G is said to be a *bridge* if each path from x to y in G passes through edge (x, y).

Prove that an edge is a bridge if and only if it does not lie on any simple cycle of G.

If $e = (x, y)$ is a bridge edge then e cannot lie on a cycle, because if on contrary e lies on a cycle, say C, then $C \setminus e$ gives an $x - y$ path not passing through e.

If $e = (x, y)$ is not a bridge edge in G, then there exists a path from x to y, say P, not passing through e, so $(P \cdot e)$ gives a simple cycle containing e.

(c) Device an $O(m+n)$ time algorithm that verifies whether a given directed graph with n vertices and m edges is strongly connected (i.e. has exactly one SCC) using only BFS traversal.(Also provide short correctness argument).

2 strings: Convert str1 → str2 LCS

str1 & str2 op^n allowed

Insert
Remove
Update

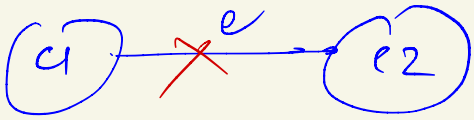
→ Remove A - LCS

Add B's

Total edits A-LCS + B-LCS

A + B - 2 LCS

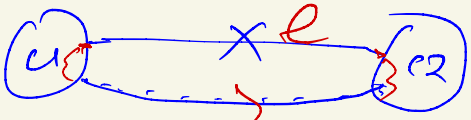
(b)



To prove :- bridge doesn't lie on a cycle.

→ See if e is bridge lying on a cycle.

Do c/e →

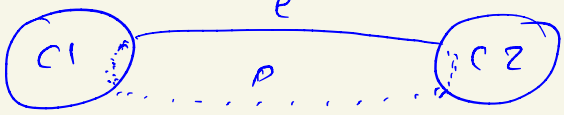


We still get c1, c2 connected by an edge that was part of the cycle.

We get a path from c1 to c2, not containing e

AND

If e is not bridge :- Then there will be



a path P s.t. P + e = cycle.

$O(m+n)$ algo using only BFS

n → vertices
m → edges

whether, SCC or not.

SCC → ∀ pair (x, y) $\quad x \rightsquigarrow y$, $y \rightsquigarrow x$

(1) Pick a node y → BFS(G) should have n vertices

(2) Cal. G^T → edges reversed

(3) BFS(G^T) → should still have all vertices.

→ DFS → Connected Graph

SCC or weakly

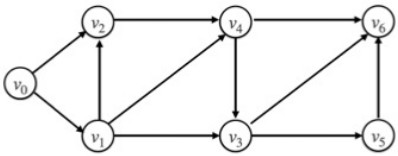
$x \rightsquigarrow y$ ✗

Take a vertex s, compute BFS from s in G and G^R (graph obtained by reversing directions). If both contains n vertices then report G has a unique SCC, else G have multiple SCCs.

(d) Device the most efficient algorithm to verify whether a given directed graph with n vertices and m edges has a unique topological ordering. (Also provide a very short correctness proof of your algorithm).

Compute any topological ordering of G using DFS traversal, let this be $(v_1, \ldots, v_n)$. Now verify that each (v_i, v_{i+1}) pair is an edge. If not, then we can swap them to obtain a new topological ordering. If each (v_i, v_{i+1}) pair is an edge in G then we report that it has a unique topological ordering, otherwise G has multiple topological ordering.

(e) Draw a DFS tree of the graph below with respect to source v_0. Also provide start-time and finish-time of all the vertices with respect to your DFS tree. Compute a topological ordering of the graph using the finish-time of vertices with respect to the DFS tree compute by you.



One of DFS trees is $(v_0 \to v_1 \to v_2 \to v_4 \to v_3 \to v_5 \to v_6)$. The start times are $1, 2, 3, 4, 5, 6, 7$, and finish times are $14, 13, 12, 11, 10, 9, 8$.

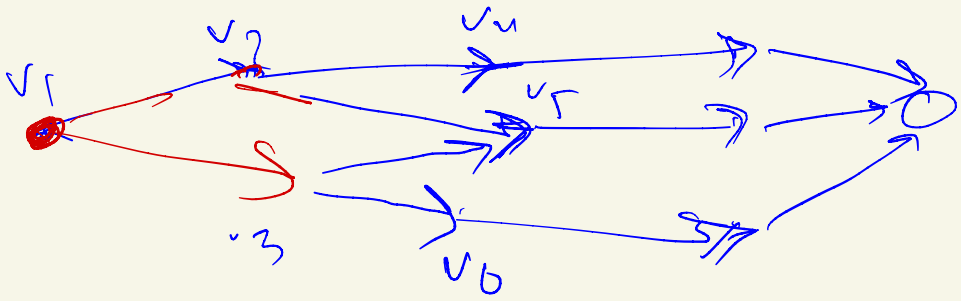
Also $(v_0, v_1, v_2, v_4, v_3, v_5, v_6)$ is a topological ordering since these are vertices listed in the decreasing order of their finish times.

(There can be other possible DFS trees and topological ordering.)

2 Gemstones [6 marks]

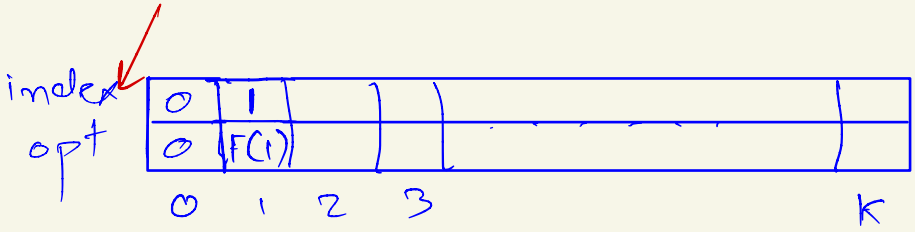
You are given set of k precious gemstones and a function $F : \mathbb{N} \to \mathbb{N}$ where $F(i)$ denotes the market price of a box of i gemstones (for $1 \leq i \leq k$). Your task is to find the optimal way to partition k gemstones into smaller sets that on selling provides you maximum overall profit.

Design an $O(k^2)$ time algorithm to output the optimal partitioning for k gemstones in a 'list'. Also provide a short correctness argument.



v_1 | v_2 v_3 | v_4 v_5 v_6 |

v_3 v_2 v_5 v_4 v_6



$i = 2 \rightarrow k$

$index[i] \leftarrow \underset{j \in [1 \dots i]}{\operatorname{argmax}} \{ F(j) + opt[i-j] \}$

$opt[i] \leftarrow F(index[i]) + opt[i - index[i]]$

Algorithm 1: GemstonePartition(k)

```
Compute two arrays index and opt of size k + 1 (indexed 0 to k);
Set opt[0] = 0 and index[0] = 0;
Set opt[1] = F(1) and index[1] = 1;
for i = 2 to k do
    index[i] = arg max{F(j) + opt[i − j] | j ∈ [1, i]};
    opt[i] = F(index[i]) + opt[i − index[i]];
end
Let L be an empty list;
x ← k;
while x > 0 do
    L.append(Index[x]);
    x ← x − Index[x];
end
Return(L);
```

3 Distances [$(3 + 3 + 2 + 4) = 12$ marks]

$G = (V, E, wt)$ is a directed, strongly-connected, weighted graph with n vertices and m edges, where wt is the edge-weight function. It is given that G has no cycle of negative weight. Let $s \in V$ be a vertex in G. Define a **new** weight function wt^* as follows:

$$wt^*(x, y) := wt(x, y) + dist_G(s, x) - dist_G(s, y).$$

(a) Show that for each edge $(x, y) \in E$, $wt^*(x, y) \geq 0$. Further, the values $wt^*(e)$, for $e \in E$, are computable in $O(mn)$ total time.

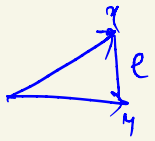
We have $dist_G(s, y) \leq dist_G(s, x) + wt(x, y)$. This implies $wt^*(x, y) \geq 0$.
To compute wt^* we can first use Bellman-Ford to compute distances from source s, and next add appropriate values to edge weights.

(b) For a path P, define $wt(P) := \sum_{e \in P} wt(e)$ and $wt^*(P) := \sum_{e \in P} wt^*(e)$.
Prove that for any $P = (v_0, \ldots, v_t)$, $wt^*(P) = wt(P) + dist_G(s, v_0) - dist_G(s, v_t)$.

$$\begin{aligned} wt^*(P) &= \sum_{i=1}^{t} wt^*(v_{i-1}, v_i) \\ &= \sum_{i=1}^{t} \left(wt(v_{i-1}, v_i) + dist_G(s, v_{i-1}) - dist_G(s, v_i) \right) \\ &= dist_G(s, v_0) - dist_G(s, v_t) + \sum_{i=1}^{t} wt(v_{i-1}, v_i) \\ &= wt(P) + dist_G(s, v_0) - dist_G(s, v_t) \end{aligned}$$

Path P

2(b) $wt(P) = \sum_{e \in P} wt(e)$

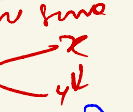


$wt^*(P) = \sum_{e \in P} wt^*(e)$

$\Rightarrow wt(e) + D_y x - D_x y$

$\in$ some time

$wt^*(P) = \sum_{i=1}^{t} wt^*(v_{i-1}, v_i)$



$= \sum_{i=1}^{t} \left[wt(v_{i-1}, v_i) + D_y(s, v_{i-1}) - D_y(s, v_i) \right]$

$\Rightarrow wt(P) + D_y(s, v_0) + D_y(s, v_1) \ldots D(s, v_{t-1})$

$- D_y(s, v_1) - D_y(s, v_2) \ldots D_y(s, v_t)$

$$\boxed{wt(P) + D_y(s, v_0) - D_y(s, v_t)}$$

$wt^A \rightarrow +ve$

Dijkstra

3(c) Argue that

$\forall (x, y) \in V$. Q is shortest from $x \to y$ acc to wt.

(iff)

Q is shortest from $x \to y$ acc to wt^*

- For any x-y path.

$$wt(Q^*) = wt(Q) + e(x, y)$$

So, a path minimizes $wt(Q^*)$ (iff) it minimizes $wt(Q)$

(d) $O(mn)$ algo to compute dist. between all vertex pairs

→ we can work with w^* instead of w

→ $w^* \geq 0$ non-ve

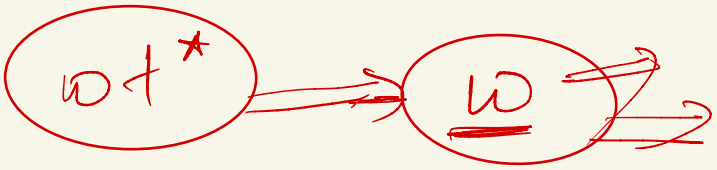
→ $O(m \log m)$

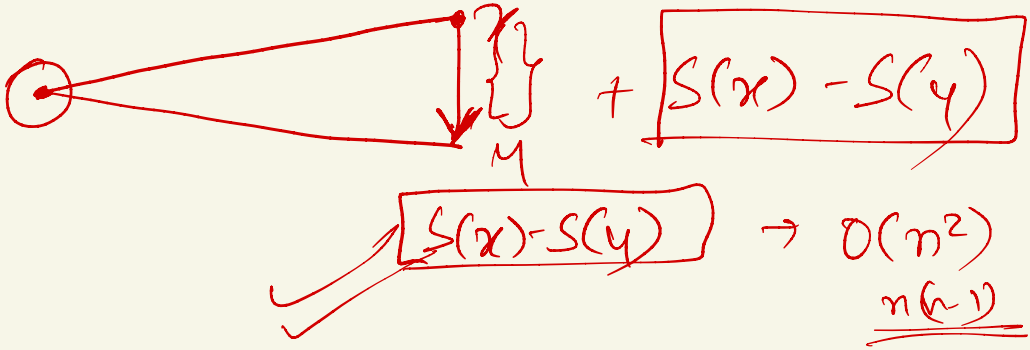
∀ vertex sum Dijkstra = $O(mn \log m)$

$\forall v \longrightarrow w+^* \rightarrow O(E \log V)$

$(m \log n)$

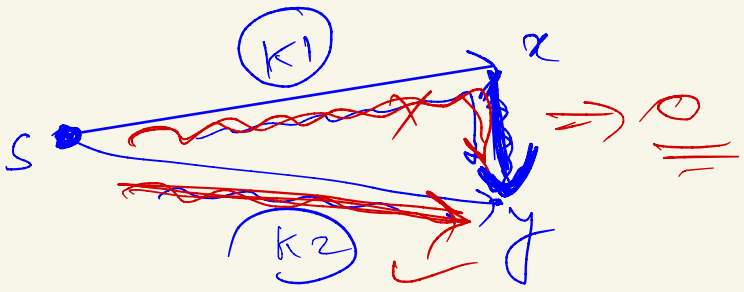
$\longrightarrow w+^{\#}$ $(nm \log n)$





dist = shortest path

3(a)



$$wt^*(x,y) = wt(x,y) + K1 - (K1 + wt(x,y))$$

$$= 0$$

OR

$$= wt(x,y) + K1 - (K2)$$

$K1 > K2$

This can only happen when

$$K1 + x > K2$$

So

$$wt^*(x,y) > 0 \Rightarrow$$

Imp: Δ inequality.

(c) Use (b), to argue that for $x, y \in V$, a path Q is a shortest path from x to y according to weight function wt if and only if Q is a shortest path from x to y according to weight function wt^*.

For any $x - y$ path Q, $wt^*(Q) = wt(Q) + c_{x,y}$, where $c_{x,y}$ is a constant (see (c)). So a path minimizes $wt^*(Q)$ iff it minimizes $wt(Q)$.

(d) Present an $O(mn \log n)$ time algorithm to compute the distance between all the vertex-pairs in the input graph G. Also provide correctness argument.

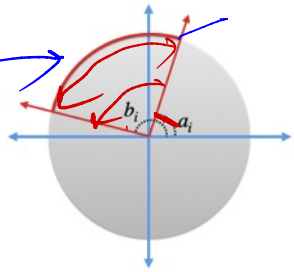
For each $x \in V$, compute a shortest-path-tree T_x from x according to weight function wt^* in $O(m \log n)$ time using Dijkstra's algorithm. So all pairs distances according to weight function wt^* can be computed in $O(mn \log n)$ time. Now it remains to add adjustments given by part (c).

We can compute wt^* as well distances from s in $O(mn)$ time by one run of Bellman ford algorithm. So the adjustment to each of n^2 pairs to obtains distance according to original weight function wt can be done in total of $O(mn)$ time.

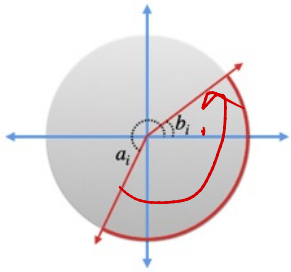
4 Covering the circumference [$6 + 3 = 9$ marks]

Let C be a unit circle on $x - y$ plane centered at the origin. You are given n arcs on C numbered $1, \ldots, n$. For $i \in [1, n]$, the arc i starts at an angle a_i from x-axis (in anticlockwise direction) and goes up to angle b_i (in anticlockwise direction).

Further, it is given to you that the union of all the arcs covers the entire circumference of the circle.



$a_i < b_i$



$a_i > b_i$

$a_i - 2\pi < b$

(a) Device an $O(n \log n)$ time algorithm that given an arc (a_{i_0}, b_{i_0}), finds a subset S of the minimum possible size such that $S \cup \{ (a_{i_0}, b_{i_0}) \}$ covers the circumference of the circle. (Also provide a correctness argument).

(b) Use (a) to device an $O(n \log n)$ time algorithm to find a subset R of arcs satisfying:

- union of arcs in R cover the circumference of the circle, and
- $|R|$ is at most one larger than the minimum number of arcs needed to cover the circumference of the circle.

(Also provide a short correctness argument).

We are given a set S of n arcs. We will show how to compute in $O(n \log n)$ time a set S_{OPT} of minimum possible size that covers the circumference. This will solve both the problems.

Sort the arcs according to starting and ending points, and discard arcs that are strictly contained in some other arcs. This takes $O(n \log n)$ time. For any $(a, b) \in S$, compute in $O(\log n)$ time the arc $\gamma(a, b) := (a^*, b^*)$ that overlaps with (a, b) and has largest possible b^*. At this point of time, if for an arc $A \in S$, $A \cup \gamma(A)$ covers the circumference then return it an exit.

Compute a digraph G with $V(G) = S$. Let $Y \subseteq V(G)$ be the set of those arcs that contain $(1, 0)$, and $X = V(G) \setminus Y$. For each arc $A \in V(G)$, add an edge $(A, \gamma(A))$ to G iff $(A, \gamma(A)) \notin X \times Y$. So, G is an n vertex directed graph with out-degree at most 1.

Claim 1: G is a DAG.

Proof: Assume on contrary G has a cycle $C = (A_1, A_2, \cdots, A_{k+1} = A_1)$ such that for $i \leqslant k$, $(A_i, A_{i+1}) = (A_i, \gamma(A_i)) \notin X \times Y$. Cycle C must contain a vertex from Y as well as X. Let i_0 be largest index such that A_{i_0} lies in X, then, A_{i_0} has out-degree 0, as G contains no edges from set $X \times Y$. This proves G is an acyclic graph.

Claim 2: If G is an n vertex DAG with out-degree at most 1, then we can compute in $O(n)$ total time for each $A \in V(G)$ the farthest vertex $F(A)$, and the distance $d(A)$ from A to $F(A)$.

Proof: Look at vertices of out-degree zero, and compute in linear time the IN-BFS trees.

Claim 3: Consider arcs $A_1 = (a_1, b_1), \ldots, A_k = (a_k, b_k)$ in S_{OPT} sorted according to b_i's, and let $(B_1, \ldots, B_k)$ be a sequence of arcs such that $B_1 = A_1$ and for $i \geq 1$, $B_{i+1} = \gamma(B_i)$. Then $(B_1, \ldots, B_k)$ also covers the circumference and is thus a minimum covering.

Proof: One can show using induction that $A_1 \cup A_2 \cup \cdots \cup A_i \subseteq B_1 \cup B_2 \cup \cdots \cup B_i$, for $i \in [1, k]$. This along with the fact that any minimum covering has size $k = |S_{OPT}|$ proves that $(B_1, \ldots, B_k)$ is a minimum covering.

Algorithm

1. Among all those arcs $A \in Y$ for which A overlaps with $F(A)$ compute one (say A_0) for which $d(A_0)$ is minimum.
2. Return arcs on A_0 to $F(A_0)$ path.

Correctness: Let $A \in S_{OPT}$ be arc with least b_i. The correctness follows from Claim 3 and facts:
– The arc A lies in the set Y and overlaps with $F(A)$.
– For all $B \in Y$ that overlaps with $F(B)$, $d(B) \geq d(A)$.

A: a_1 a_2 a_3 ... $a_i < a_j$ ... a_n ($n \log n$)

B: b_1 b_2 b_3 ... $b_i < b_j$ b_n

$O(n \log n)$

$a_1 < b_1$

$a_2 < b_2$

$0 \le a_i, b_i \le 2\pi$

$[a_1, b_1]$ ... $[a_n, b_n]$

LHS RHS

$W = \{[a_1, b_1] \dots [a_n, b_n]\}$

W' = whenever $a_i > b_i$

$(a_i - 2\pi, b_i)$

$a_i \le b_i$ ← of ALGO needs

discard A_1

A_1 A_2

a_i a_j b_i b_j 1,0

index = k

a: 5 2 j i

b: 5 2 i j

Delete i

data structures where delete takes $O(\log n)$ time.

$a_j < a_i$

$b_i < b_j$

Now $\forall$ [a, b] compute [$\gamma(a,b)$]

$\Rightarrow$ the are ourlapping with (a,b) and highest b^*

So, now $A \cup \gamma(A)$

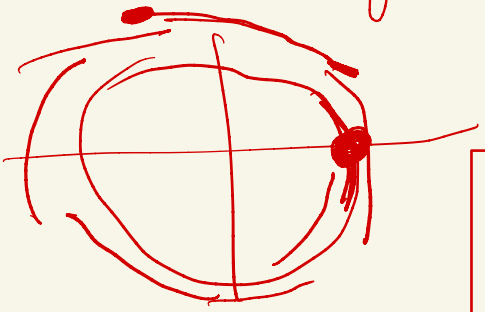
$A - \gamma(A) - \gamma(\gamma A) - \cdots$

$(a,b) \Rightarrow w$

Digraph (G) — {ai; bi}

Vertex Set ⇒ S — all are

$\gamma \in V$ → set of vertices having (1,0)



iff (A, γ(A)) ∈ X × Y

$X = V(G)$

$\forall A \in V(G)$ (A, γ(A))

This graph would be a n-vertex directed graph

out-degree almost - 1

$A \to \gamma(A) \to \gamma(\gamma(A))$

G is a DAG

- Assume G has a cycle.

$A_1, A_2 \dots A_k, A_1, \gamma(A_k)$

$(A_k, A_1) \Longrightarrow A_k \gamma(A_k)$

$A_k, \gamma(A_k) \notin X \times Y$

⇒ We also know that cycle must've a vertex from Y and X.

Let (b) be that index X (biggest)

Given it's biggest index

⟶ outdegree = 0

So, G has no edges from X to Y

So, G is acyclic.

Claim - 2 ⟺

G → n vertex DAG, with outdegree atmost 1.

We can compute in O(n) time ∀ A ∈ V(G), farthest vertex F(A) (&) distance d(A) (A → F(A))

→ can use BFS to calculate compute shortest-path trees!

If outdegree 1

edges going towards parent.

Look at outdegree → 0 → and compute shortest path trees.

have to look at leaf-to-root path of smallest length

CLAIM 3: $A_1=(a_1,b_1)$ $A_2=(a_2,b_2)$ ... (a_K,b_K) arcs sorted according to b_i's. in SORT

$(B_1, B_2 \dots B_K)$ be a sequence such that

$B_1 = A_1$ $\quad B_{i+1} = \gamma[B_i]$

A_1 $\gamma(A_1)$ $\gamma(\gamma(A_1))$ ....

Then, it covers the circumference

→ Induction → to show that

$A_1 \cup A_2 \dots A_i \subseteq B_1 \cup B_2 \dots B_i$ $i \in [1..K]$

$\gamma(A)$ $\gamma(\gamma(A)) \dots$ ∞A

DAG

ALGORITHM :-

Y → leafs F(A) ⇒ root

⇒ ∀ A ∈ Y overlapping with F(A)

compute f_{A_0} s.t. $d(A_0)$ is minimum.

→ Return Arcs on $A_0 \rightarrow F(A_0)$ path

∀ A → leaf nodes

(2) one at leaf & root should overlap.

If $A_1 - A_2 - \cdots A_k$

Instead of A_2 can have $\gamma(A_1)$

$A_1 \quad \gamma(A_1) \quad \gamma(\gamma A_1)$

GEMSTONES

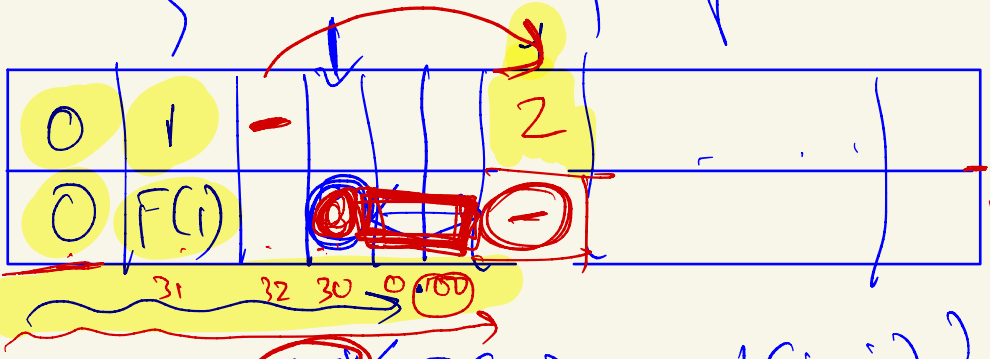
K → gemstones

F : N → N

F_i : denotes market price of box of i gemstones.

→ Find optimal way to partition K gemstones in smaller sets

selling → max profit



index[i] ← $\max_{j=1}^{i} \{ F(j) + opt(i-j) \}$

j ∈ [1 ... i]

F(j) + OPT(diff)

F(2) + OPT(3 - 1)

$opt[i] = F(index[i]) +$

$opt[i - index[i]]$

How much you earn by optimal partitioning.