Contest – 2

Dyikstra Complexity O(Vlog2 (V) + E), Bellman O(VE) or O(V^3)

Problem – C

Assign the distances to the values of train route instead of infinity in the beginning. Count when the train routes are relaxed. To find when these routes are relaxed check if the node is unvisited and has a particular value. Finally return the count.

Problem – D

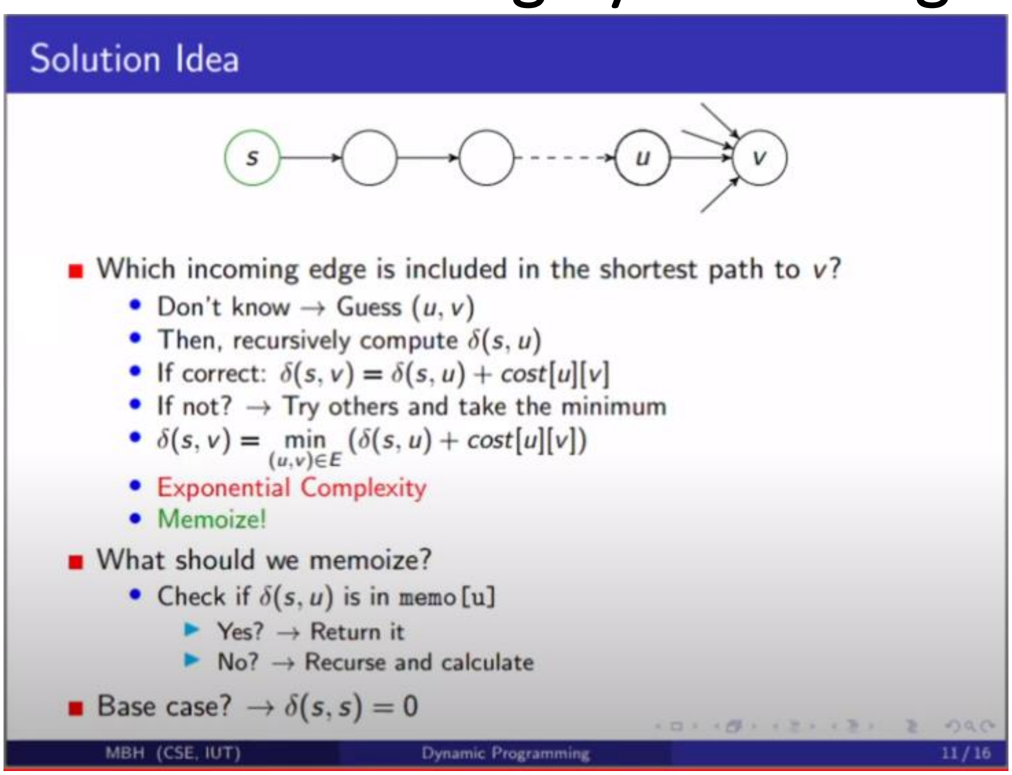
The graph is to be relaxed using Bellman Ford. We chose Bellman Ford instead of Djikstra because Bellman Ford uses

Problem – E

Typical Djikstra is used in this case but the relaxation condition has been changed. We need to relax when the maximum weight in our path (set of weights) will be minimum among all the possible paths. To do so, we store the maximum weight in a path and relax whenever the maximum weight is greater than the updated weight we found.

If (dist(v) > max(v_weight, dist(u)) dist(v) = max(v_weight, dist(u);

Shortest Path using Dynamic Programming



Relate Subproblems

- State the topological order$\rightarrow$ Guess if required
- Argue the relations are acyclic
- Form a DAG

Identify Base Cases

- Solutions for independent subproblems

Examples

- Fibonacci Series: $F(1) = F(2) = 1$
- Shortest Paths:
  - $d(s) = \delta(s, s) = 0$ and $d(v) = \infty$ for all $v \neq s$ without incoming edge
  - $d(s, 0) = 0$ and $d(v, 0) = \infty$ for all $v \neq s$ without incoming edge

Complexity

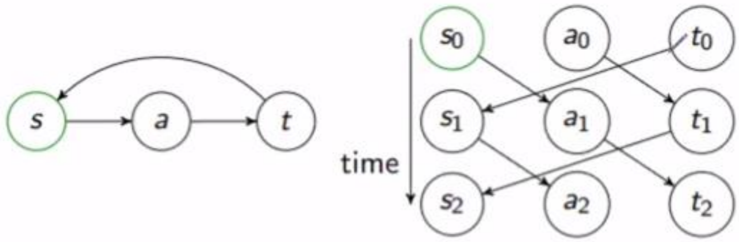
- $\#$ of subproblems $\times$ Time/subproblem
- Subproblems$\rightarrow$ $\delta(s, v)$
- $\#$ of subproblems$\rightarrow$ V
- Time for subproblem $\delta(s, v)$
  - $indegree(v)$ $\rightarrow$ Not equal for all subproblems
  - Cannot take direct product here $\rightarrow$ Sum over all subproblems
  - Calculate min each time $\rightarrow$ $\Theta(1)$
  - Total: $indegree(v) + 1$
- Total Time $\sum_{v \in V} (indegree(v) + 1)$

$$= \sum_{v \in V} indegree(v) + \sum_{v \in V} 1$$
$$= \Theta(V + E)$$

- But that's TopSort + One Pass Relaxation!

Shortest Path in Cyclic graphs

Cyclic Graphs



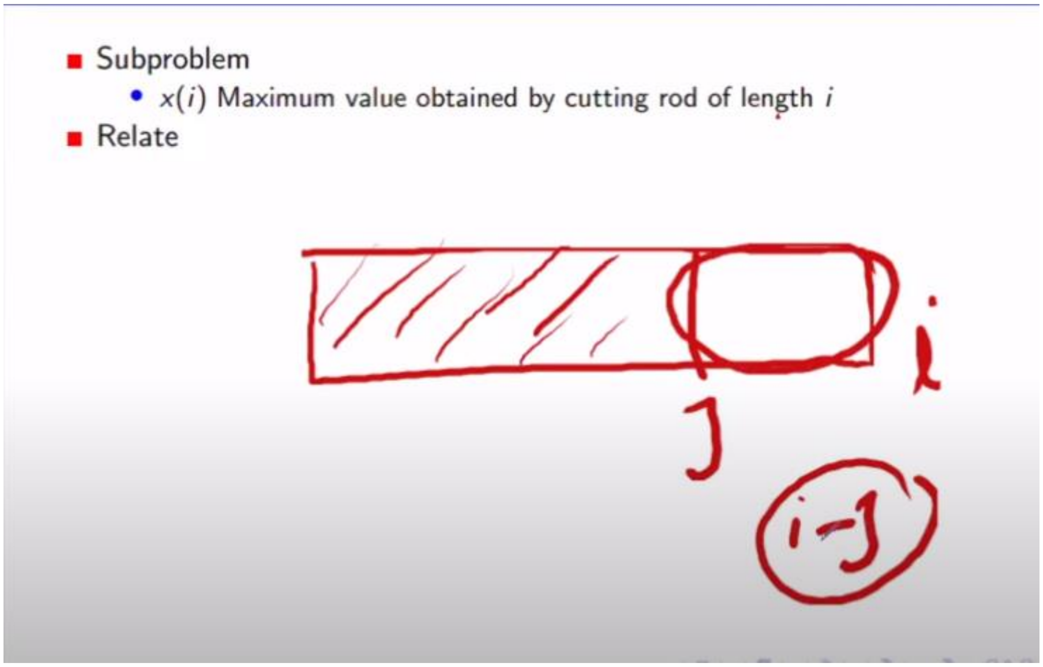
- Simple path will take $(v-1)$ edges
- Make $k=(v-1)$ copies of the graph
- Edge from current layer to next layer
- $\delta_k(s,v) =$ Weight of the shortest path from s to v using $\leq k$ edges
- General Form: $\delta_k(s,v) = \min\limits_{(u,v)\in E} (\delta_{k-1}(s,u) + cost[u][v])$
- Goal: $\delta_{|V|-1}(s,v)$

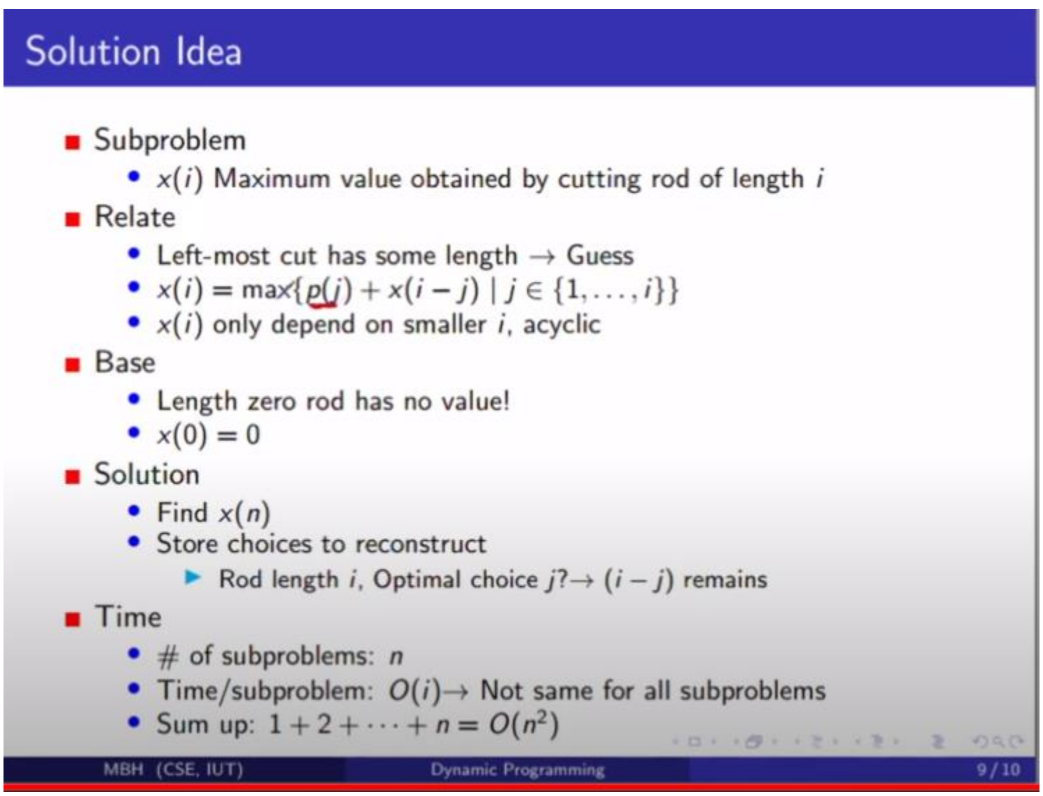
Complexity

- $\#$ of subproblems $\times$ Time/subproblem
- Subproblems $\rightarrow \delta_k(s,v)$
- $\#$ of subproblems
  - Value of $k \rightarrow [0, V-1]$
  - For each $k \rightarrow v \in V$
  - Total: V^2
- For each k, time for subproblem, $\delta(s,v)$
  - $indegree(v) \rightarrow$ Not equal for all subproblems
  - Cannot take direct product here $\rightarrow$ Sum over all subproblems
  - Calculate min each time $\rightarrow \Theta(1)$
  - Total: $indegree(v)+1$
- Total time for each $k \rightarrow \Theta(V+E)$
- How many ks? $\rightarrow [0, |V|-1] \rightarrow |V|$
  - Total: $\Theta(V \cdot E)$

ROD CUTTING PROBLEM

In relate you must mention that the values are stores in memory array





Longest Common Subsequence

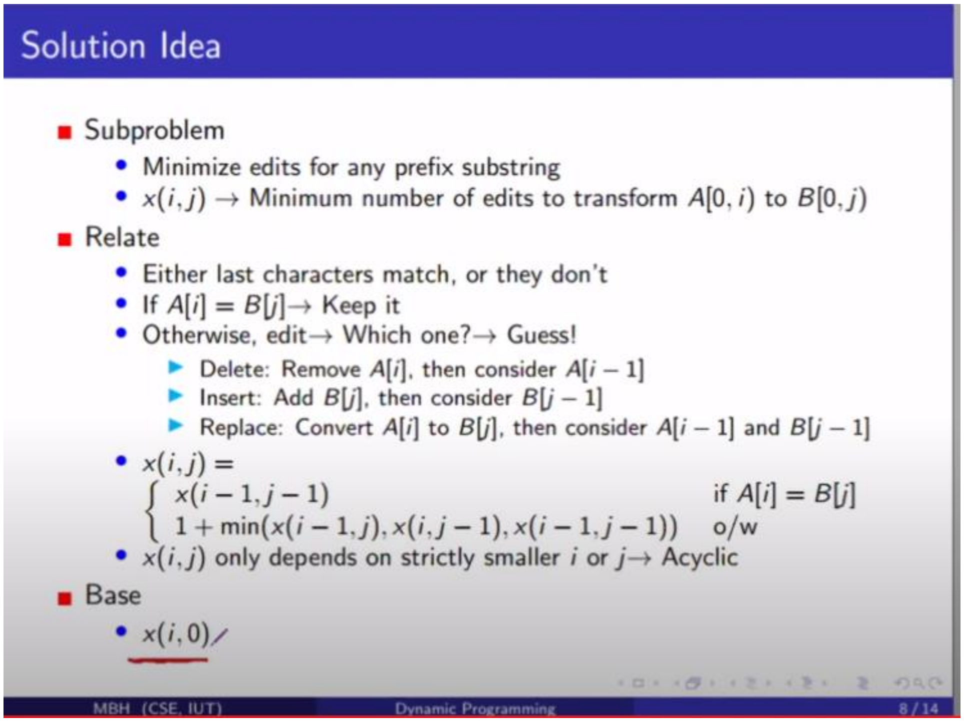
Solution Idea

132
126
103

- Subproblem
  - Maximize the length of subsequence of prefixes
  - $x(i,j) \to$ Length of LCS of prefixes $A[0,i)$ and $B[0,j)$
- Relate
  - Either last characters match or they don't $\to$ Guess!
  - If last characters of A and B match
    - some LCS will use them
    - Only improves the solution
  - If they don't match
    - Both cannot be in the LCS
    - One of them might be $\to$ Try both
  - $x(i,j) = \begin{cases} x(i-1,j-1)+1 & \text{if } A[i-1] = B[j-1] \\ \max\{x(i-1,j), x(i,j-1)\} & \text{o/w} \end{cases}$
  - $x(i,j)$ only depends on strictly smaller i or j $\to$ Acyclic
- Base
  - No LCS if one string is empty $\to$ $x(i,0) = x(0,j)$

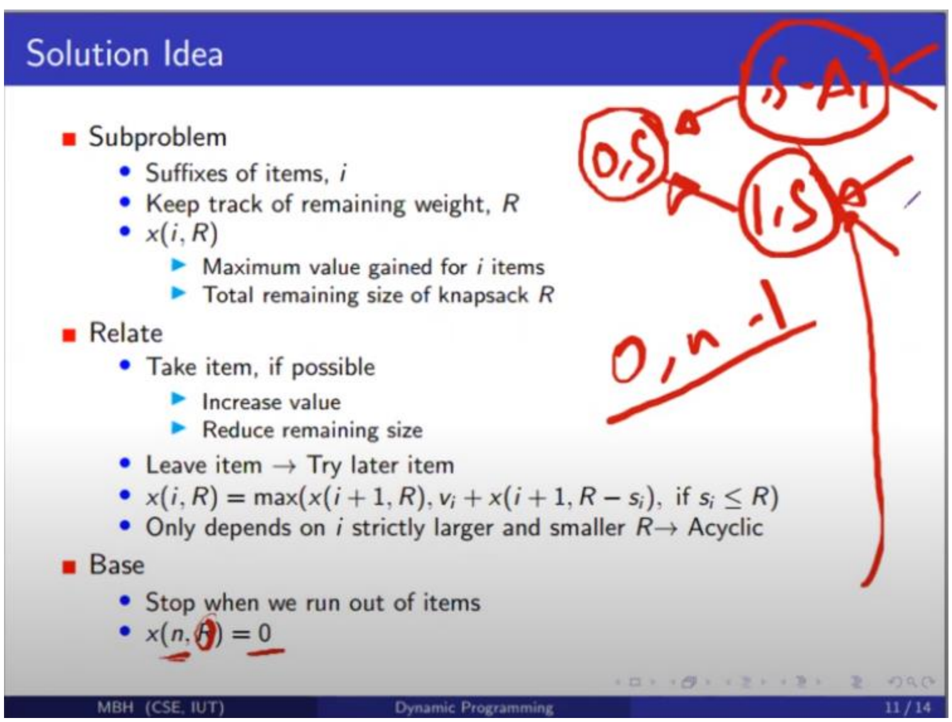
- Solution
  - Find $x(|A|,|B|)$
  - Store parent pointers
    - Add letter to subsequence if both i and j decrease
- Time
  - \# of subproblems: $(|A|+1)(|B|+1)$
  - Work per subproblem: $O(1)$
  - Running time: $O(|A||B|)$

Edit Distance

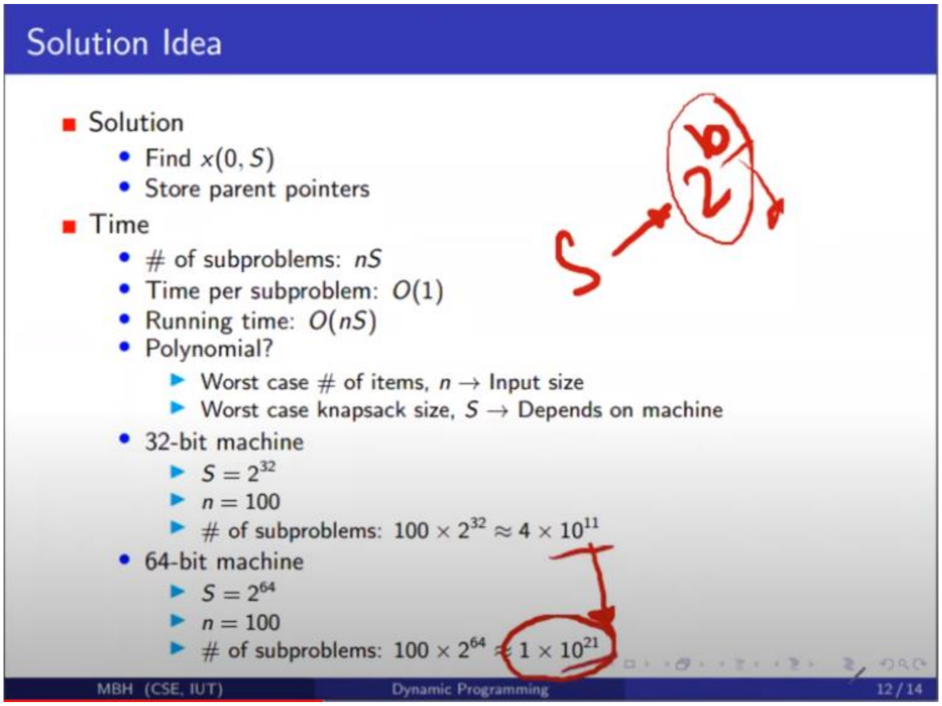


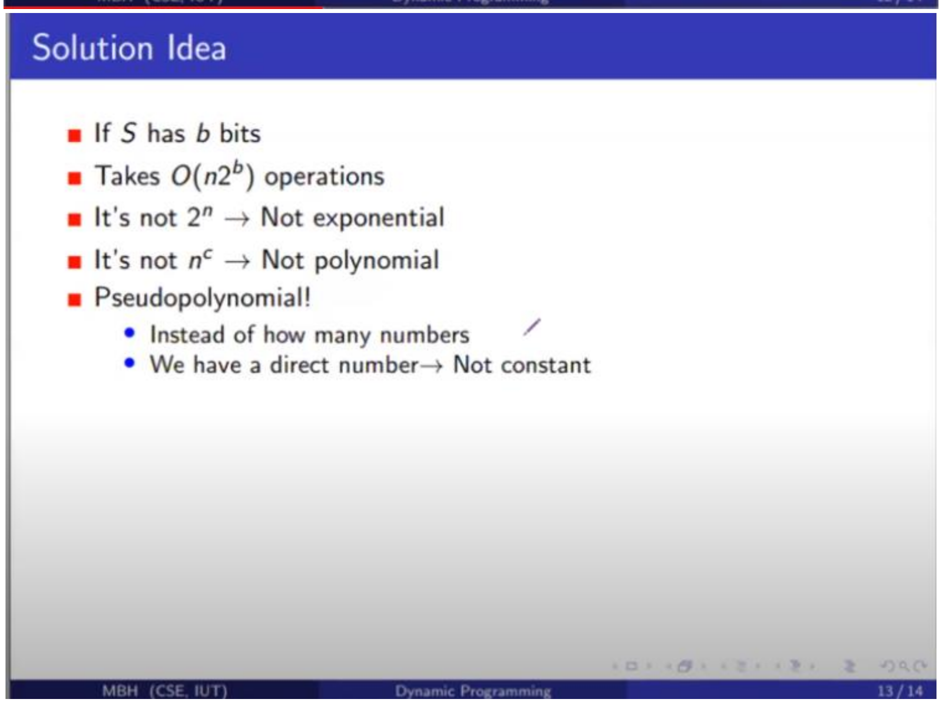
Solution and time is same as previous one

Knapsack



Si > R then x(I,R) = x(i+1,R)





Capsicum/Boredom: Adjacency changing Problem

```
mem[i] = max(mem[i] + mem[i-2], mem[i-1]);
```

Subset Sum

Solution Idea

- Subproblem
  - Suffix of elements, check for smaller sum
  - $x(i, s)$ = Is there a subset $S \subseteq A[i:]$ with $\sum(S) = s$
- Relate
  - Whether $A[i] \in S \rightarrow$ Guess
  - Picking would reduce the value of s
  - $x(i, s) = OR(x(i+1, s), x(i+1, s - A_i)$ if $A_i \leq s)$
  - $x(i, s)$ only depends on larger i or smaller $s \rightarrow$ Acyclic
- Base
  - $x(n, 0) =$ `True`
  - $x(n, > 0) =$ `False`
- Solution
  - Find $x(0, t)$
- Time
  - # of subproblems: $(n+1)(t+1)$
  - Work per subproblem: $O(1)$
  - Running time: $O(nt) \rightarrow$ Pseudopolynomial

MBH (CSE, IUT) Dynamic Programming 3 / 11

Submultiset Sum (coin exchange)

Solution Idea

- Subproblem
  - Make part of sum, using suffix of elements
  - $x(i, s)$ = minimum items required to make s
- Relate
  - Take m_i items $\rightarrow$ Reduce the target by $m_i \times A_i$
  - Can't take any amount for $m_i \rightarrow m_i \times A_i \leq s$
  - $x(i, s) = \min(m_i + x(i+1, s - m_i \times A_i) | m_i \in [0, \frac{s}{A_i}])$
  - $x(i, s)$ only depends on larger i or smaller $s \rightarrow$ Acyclic
- Base
  - $x(n, 0) = 0$
  - $x(n, > 0) = \infty$
- Solution
  - Find $x(0, t)$
- Time
  - # of subproblems: $O(nt)$
  - Work per subproblem: $O(t)$
  - Running time: $O(nt^2)$

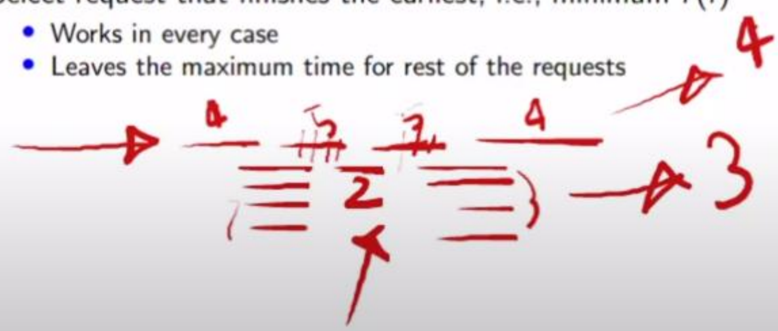
$m_i \rightarrow \frac{s}{A_i}$ $\quad \frac{t}{1} = t$

MBH (CSE, IUT) Dynamic Programming 5 / 11

Divide and Conquer

Possible Rules

- Select request that starts the earliest, i.e., minimum $s(i) \rightarrow \times$
- Select request that is the smallest, i.e., minimum $f(i) - s(i) \rightarrow \times$
- Select request with the minimum number of incompatible requests $\rightarrow \times$
- Select request that finishes the earliest, i.e., minimum $f(i)$
  - Works in every case
  - Leaves the maximum time for rest of the requests



Small Change

- Each request has weight $w(i)$
- **Goal:** Select a compatible subset of requests with maximum weight

Dynamic Programming

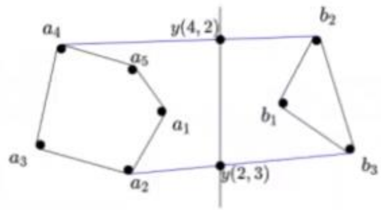
- Subproblem: $x(i) =$ Maximum weight gained by starting from task i
- Relate: $x(i) = w(i) + \max(x(j)$ where $s(j) \geq f(i)\,\forall j)$
- Base: If i is the last task, $x(i) = w(i)$
- Solution: $\max\limits_{1 \leq i \leq n}(x(i))$
- Time: $O(n^2)$ → $O(n \log n)$
- Better?
  - Sort by start time
  - For each request i
    - Choose it as first and check for the immediate next j for which $s(j) \geq f(i)$, or
    - Leave it and check the immediate next j for which $s(j) > s(i)$
    - Take the best of the two

      $\rightarrow x(i) = \max(w(i) + x(j)$ where $s(j) \geq f(i), x(j)$ where $s(j) > s(i))$

$O(n)$

$O(n)$ $O(1)$

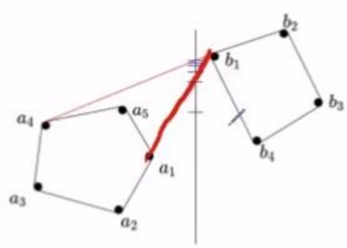
How to Combine - Finding Tangents



n

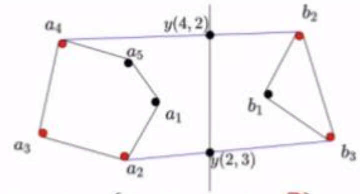
- Two convex hulls A and $B \rightarrow$ Need to merge
- Brute force: Generate pairwise (from A to B) segments and check
- a_4, b_2 is called the upper tangent $\rightarrow$ Max $y(i,j)$
- a_2, b_3 is called the lower tangent $\rightarrow$ Min $y(i,j)$
- Complexity: $O(n^2)$

How to Combine - Finding Tangents



- Picking the max y for both is not enough
- Need to maximize $y(i,j)$
- Two-Finger Algorithm
  - Start with line segment using rightmost of a and leftmost of b
  - Move clockwise for $b \rightarrow$ Update if it improves $y(i,j)$
  - Move anti clockwise for $a \rightarrow$ Update if it improves $y(i,j)$
  - Repeat until $y(i,j)$ converges
- Similarly find lower tangent

How to combine - Merge Two Lists



$$a = \{a_1, a_2, a_3, a_4, a_5\}$$
$$b = \{b_1, b_2, b_3\}$$

$a_4\ b_2\ b_3\ a_2\ a_3$

- Found two tangents? Merge.
- Cut and Paste Method
  - Start from left of upper tangent
  - Go to the right of upper tangent
  - Keep going clockwise until you reach right of lower tangent
  - Move to left of lower tangent
  - Keep going clockwise until you reach left of upper tangent

Complexity

- Sort $\rightarrow O(n \log_2(n))$ (e.g. Merge Sort) or $O(n)$ (e.g. Radix Sort)
- Divide and Conquer
  - Divide
    - Choose partition at the center
    - $T(n) = 2T\left(\frac{n}{2}\right)$
  - Merge
    - Two finger algorithm $\rightarrow O(n)$
    - Merge two lists $\rightarrow O(n)$
  - Total
    - $T(n) = 2T\left(\frac{n}{2}\right) + O(n) \rightarrow$ Merge sort
    - $T(n) = O(n \log_2(n))$
- Total Complexity $\rightarrow O(n \log_2(n))$

$O(n^3)$

Divide and Conquer Approach

- Assume $n = 2^k$(Or make it, by appending zeros)
- Divide the matrix in four $\frac{n}{2} \times \frac{n}{2}$ matrices
- Continue dividing until we get single element→ Use brute force
- Combine

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}, B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}, C = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix}$$

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ C_{12} &= A_{11}B_{12} + A_{12}B_{22} \\ C_{21} &= A_{21}B_{11} + A_{22}B_{21} \\ C_{22} &= A_{21}B_{12} + A_{22}B_{22} \end{aligned}$$

- Cost
  - Multiplications of size $\frac{n}{2} \rightarrow 8$
  - Addition Count: $4 \times \frac{n^2}{2}$
  - $T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2) = \Theta(n^3)$