COL 351: Analysis and Design of Algorithms

Lecture 17

Tutorial Problems

- Covid-test centres (4.5)
- Longest Common Subsequence (4.2)
- Bipartite graph (3.3)
- Delay Synchronisation (2.4)

Covid-test centres

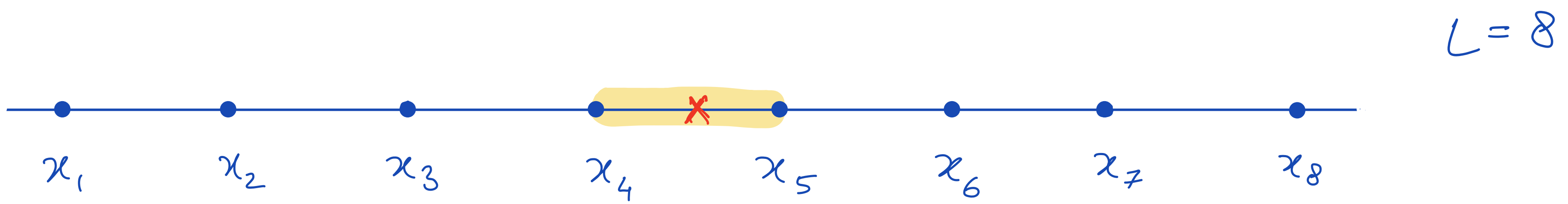
Given: A sorted array A of locations of n residents, and a parameter k.

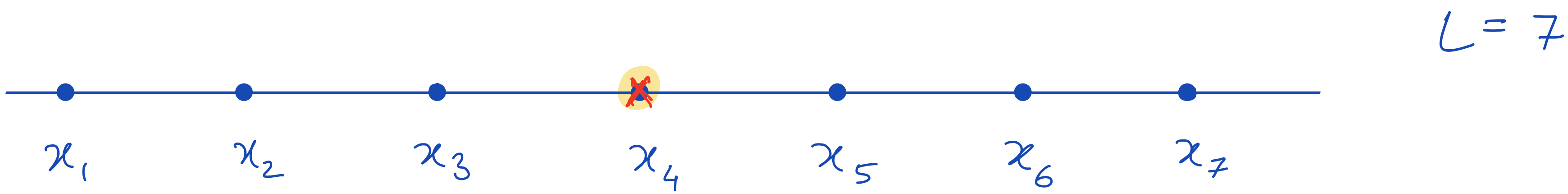
Goal: Compute a k-sized integer array C of locations of Covid-test centres that minimizes:

$$\sum_{i=1}^{n} d_i, \quad \text{where,} \quad d_i = \min_{j \in \{1,\ldots,k\}} \left| A[i] - C[j] \right| .$$

What if we have to open only one test centre?

Lemma: If we want to just open one test centre for residents at locations $x_1, \ldots, x_L$ sorted in non-decreasing order then the best choice is any point lying between $y = x_{\lfloor \frac{1+L}{2} \rfloor}$ and $y = x_{\lceil \frac{1+L}{2} \rceil}$.



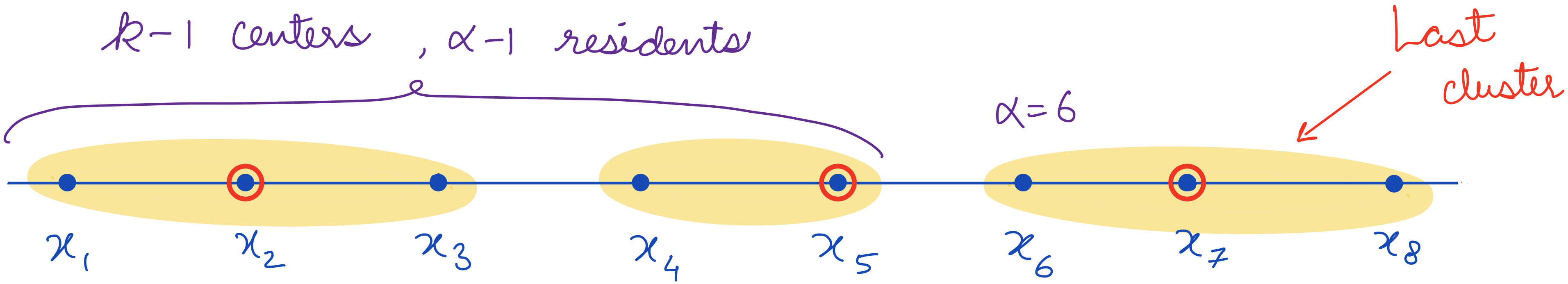


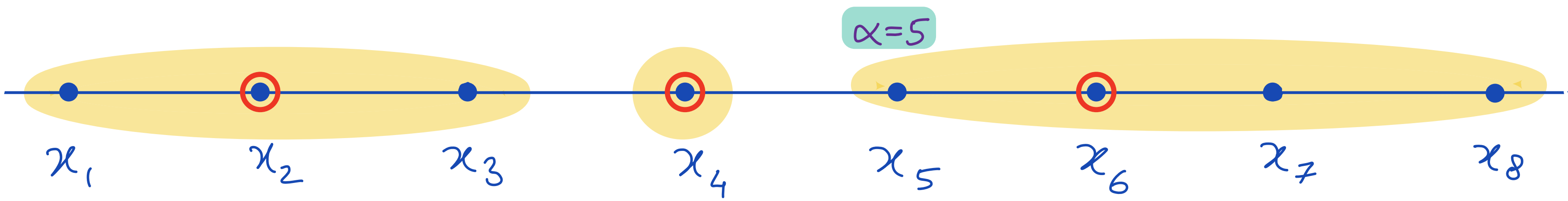
Covid-test centres: dynamic program

Make table T of size $n \times k$.

$T[i, j]$: Optimal solution for first i residents in A with j testing centres.

Eg. $n = 8$ $k = 3$





Covid-test centres: dynamic program

Make table T of size $n \times k$.

$T[i,j]$: Optimal solution for first i residents in A with j testing centres.

$$T[i,j] \;=\; \min_{\alpha \in [1,i]} \left(T(\alpha - 1,\, j-1) \;+\; \sum_{r=\alpha}^{i} \left| A\left[r\right] - A\left[\frac{\alpha + i}{2}\right] \right| \right)$$

Longest Common Subsequence

Given: Two sequences $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_n)$.

Goal: Compute LCS of X and Y in $O(m + n)$ space.

Longest Common Subsequence

$X_i = (x_1 \ldots x_i)$
$Y_j = (y_1 \ldots y_j)$

Given: Two sequences $X = (x_1, \ldots, x_m)$ and $Y = (y_1, \ldots, y_n)$.

Goal: Compute LCS of X and Y in $O(m + n)$ space.

1. Create a 2D-array "T" of size $(m+1) \times (n+1)$.
2. For $i = 0$ to m : $T[i, 0] = 0$
3. For $j = 0$ to n : $T[0, j] = 0$
4. For $i = 1$ to m :
   - For $j = 1$ to n :
     - If $(x_i = y_j)$: $T[i, j] = T[i-1, j-1] + 1$
     - Else $T[i, j] = \max(T[i-1, j], T[i, j-1])$

Time $= O(mn)$ Space $= O(\min\{m, n\})$ ← we can modify this algorithm to get

LCS algorithm

1. If $(x_i = y_j)$: return LCS$(X_{i-1}, Y_{j-1}) \cdot x_i$

2. ℓ_1 = Len-LCS(X_i, Y_{j-1})
3. ℓ_2 = Len-LCS(X_{i-1}, Y_j).

Will take $O(m+n)$ space, and also vacate the space.

4. If $\ell_1 \geqslant \ell_2$: return LCS(X_i, Y_{j-1})

   Else return LCS(X_{i-1}, Y_j)

LCS(X_i, Y_j)

Eg:

$m=5,\ n=3$

X_0	Y_0
X_1	Y_1
X_2	Y_1
X_2	Y_2
X_3	Y_3
X_4	Y_3
X_5	Y_3

This stack uses $O(m+n)$ space

Either m/n decreases

Computing len-LCS

$O(m+n)$ space

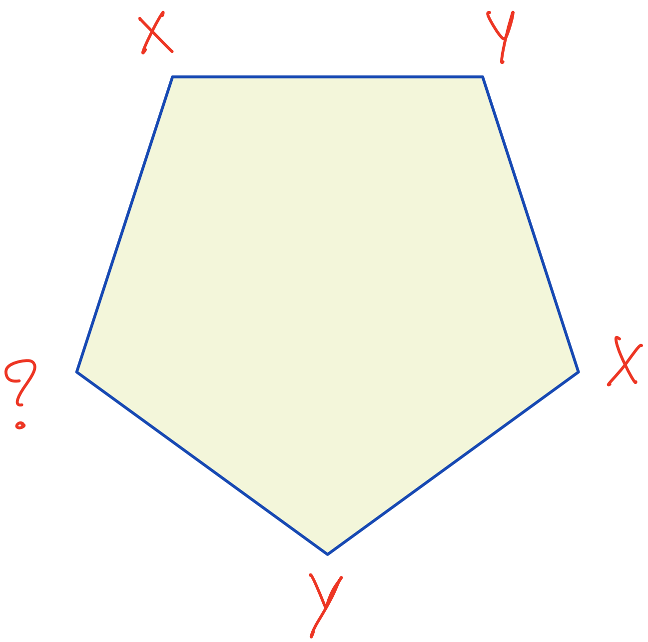
space $= O(m+n)$

Time $= O((m+n)\cdot(mn))$

Bipartite graph

Definition: A graph whose vertices can be partitioned into two sets $X,\ Y$ such that each edge edge lie in set $X \times Y$.

Claim 1: A graph G with an odd cycle cannot be bipartite.



Let $C = (a_1 \dots a_{2i+1})$ be a cycle

$a_1\ a_3\ a_5 \dots a_{2i+1} \leftarrow$ Set X

$a_2\ a_4\ a_6 \dots a_{2i} \leftarrow$ Set Y

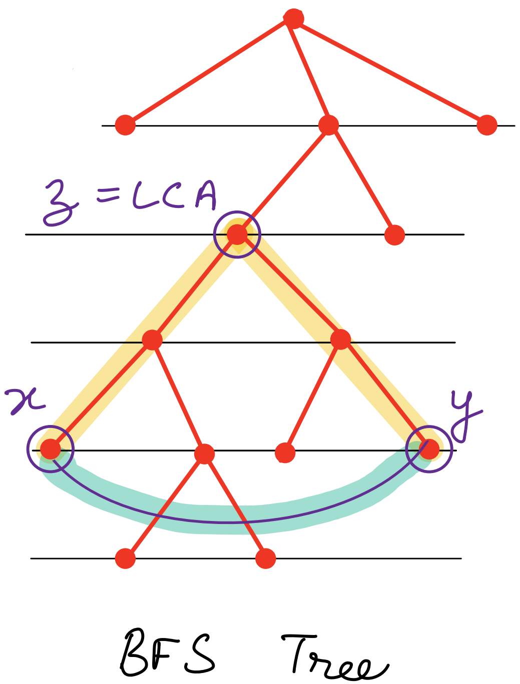
$\Rightarrow a_1$ & a_{2i+1} both lie in X

Bipartite graph

Definition: A graph whose vertices can be partitioned into two sets $X,\ Y$ such that each edge edge lie in set $X \times Y$.

Claim 2: Let G be a connected graph, and T be a BFS tree of G. Then, $\underbrace{G \text{ is bipartite}}_{S_1}$ iff $\underbrace{\text{all edges } (x, y) \in E \text{ satisfies } |\,level(x) - level(y)\,| = 1}_{S_2}$.

To show : $\neg S_2 \Rightarrow \neg S_1$



Proof: suppose S_2 fails.

Let $(x,y) \in E$ be such that $level(x) = level(x) = i$

Let $z = LCA(x,y)$ be lowest common ancestor of x, y

Then $(z \rightsquigarrow x) \cdot (x,y) \cdot (y \rightsquigarrow z)$ is odd cycle

Bipartite graph

Definition: A graph whose vertices can be partitioned into two sets $X,\ Y$ such that each edge edge lie in set $X \times Y$.

Claim 2: Let G be a connected graph, and T be a BFS tree of G. Then, $\underbrace{G \text{ is bipartite}}_{S_1}$ iff $\underbrace{\text{all edges } (x, y) \in E \text{ satisfies } |level(x) - level(y)| = 1}_{S_2}$.

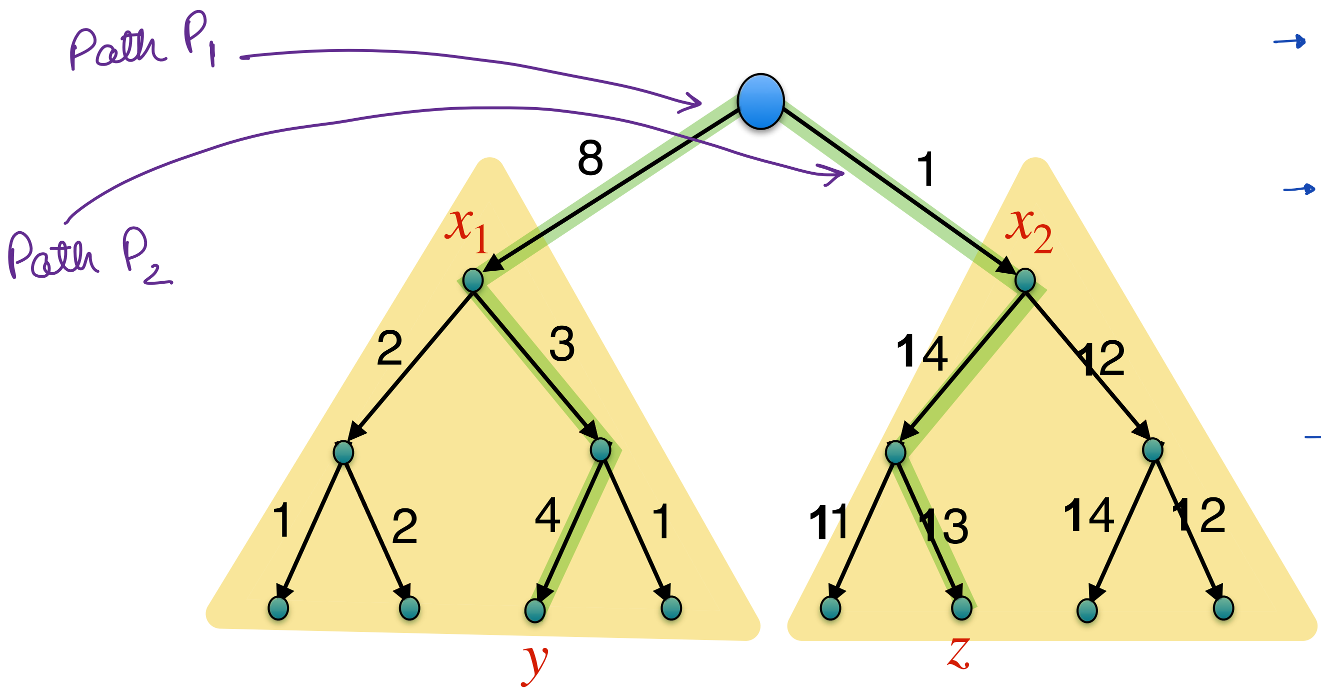
To show : $S_2 \Rightarrow S_1$

Proof: Put vertices in odd level in X
& vertices in even level in Y.

Delay Synchronisation

Given: A binary tree T with n leaves, such that each edge e has delay d_e.

Goal: Add minimum delay addition to edges of T so that delay at leaves is synchronised.



→ Solve subtree at left child x_1

→ Solve subtree at right child x_2

→ Show that in opt solⁿ of subtree x_i, the path P_i is unaffected.

→ To find opt of main tree it suffices to put $|wt(P_1) - wt(P_2)|$ amount of weight on one outgoing edge of root.