(*Ragesh Jaiswal*)

The following main topics will be covered in the course:

Graph Algorithms	*Model a problem as a graph problem and use known graph algorithms.*
Greedy algorithms	*Being greedy works! ... but only for certain problems.*
Divide and conquer	*As the name suggests "Divide and Conquer". Divide the problems into smaller sub-problems, solve them and combine their solution.*
Dynamic Programming	*Divide the problem into overlapping subproblems of various sizes. Use solution of smaller sub-problems to build solution of larger ones.*
Network Flow	*Reduce problem to a "Network Flow" problem. Solve network flow using an algorithm that builds better and better solutions in small increments.*
Backtracking	*Incrementally attempt to build solution and "backtrack" to explore all possibilities.*
Computational intractability	*Not all problems may be solved efficiently. How do we know that a problem is hard?*

In order to motivate the relevance of the above techniques for designing efficient algorithms for useful problems, we give four problems as case studies. This an optional reading at the beginning of the course. By the end of the course, you will know techniques to design algorithms for them.

Virus spread (Graph Algorithms)

Algotown has n residents labeled $1, ..., n$. The town authorities find that a set S of k residents of the town have been tested positive for a new virus. They want to closely monitor all residents who may be at risk of having the virus. They come up with the following way to define the set of residents who may be at risk of having the virus. A resident i is said to be *resident-at-risk* if and only if at least one of the following two conditions holds:

(i) $i \in S$ (*that is, i is one of the residents who have tested positive*)

(ii) There is a sequence of residents $i, j_1, j_2, ..., j_t$ for $t > 0$ such that every adjacent pair of residents in this sequence has been in close proximity in the past two weeks and $j_t \in S$.

To create a list of all residents who are *resident-at-risk*, they have conducted a survey where resident i correctly returned a list $C[i]$ of all other residents with whom resident i was in close proximity in the past two weeks. You have been asked to compile a list of all residents who are *resident-at-risk* given n, S, and $C[1], ..., C[n]$ as input. Design an algorithm for this problem.

Hand sanitiser sharing (Greedy Algorithms)

Algotown has n residents labelled $1, ..., n$. In the midst of a virus outbreak, the town authorities of *Algotown* realise that hand sanitiser has become an essential commodity. They know that every resident in this town requires at least T integer units of hand sanitiser. However, at least $\lceil \frac{n}{2} \rceil$ residents do not have enough sanitiser. On the other hand, there may be residents who may have at least T units. With very few new supplies coming in for the next few weeks they want to implement some kind of sharing strategy. At the same time, they do not want too many people to get in close proximity to implement sharing. So, they come up with the following idea:

Try to *pair up* residents (based on the amount of sanitiser they possess) such that:

1. A resident is either unpaired or paired with exactly one other resident.
2. Residents in a pair together should possess at least $2T$ units of sanitiser.
3. The number of pairs is maximised.

Once such a pairing is obtained, the residents who are left out (that is, unpaired residents) can be managed separately. The town authorities have conducted a survey and they know the amount of sanitiser every resident possesses. You are asked to design an algorithm for this problem. You are given as input integer n, integer T, and integer array $P[1...n]$ where $P[i]$ is the number of units of sanitiser that resident i possesses. You may assume that $0 \leq P[1] \leq P[2] \leq ... \leq P[n]$. Your algorithm should output a pairing as a list of tuples $(i_1, j_1), (i_2, j_2), ..., (i_k, j_k)$ of maximum size such that (i) For all $t = 1, ..., k$, $P[i_t] + P[j_t] \geq 2T$ and (ii) $i_1, ..., i_k, j_1, ..., j_k$ are distinct.

Optimal location for testing centers (Dynamic Programming)

Algotown has n residents labelled $1, ..., n$. All n residents live along a single road. The Town authorities suspect a virus outbreak and want to set up k testing centers along this road. They want to set up these k testing centers in locations that minimises the sum total of distance that all the residents need to travel to get to their nearest testing center. You have been asked to design an algorithm for finding the optimal locations of the k testing centers.

Since all residents live along a single road, the location of a resident can be identified by the distance along the road from a single reference point (which can be thought of as the starting point of the town). As input, you are given integer n, integer k, and the location of the residents in an integer array $A[1...n]$ where $A[i]$ denotes the location of resident i. Moreover, $A[1] \leq A[2] \leq A[3] \leq ... \leq A[n]$. Your algorithm should output an integer array $C[1...k]$ of locations such that the following quantity gets minimised:

$$\sum_{i=1}^{n} D(i), \text{ where } D(i) = \min_{j \in \{1,...,k\}} |A[i] - C[j]|$$

Here $|x - y|$ denotes the absolute value of the difference of numbers x and y. Note that $D(i)$ denotes the distance resident i has to travel to get to the nearest testing center out of centers at $C[1], ..., C[k]$.

(*For example, consider* $k = 2$ *and* $A = [1, 2, 3, 7, 8, 9]$. *A solution for this case is* $C = [2, 8]$. *Note that for testing centers at locations* 2 *and* 8*, the total distance travelled by residents will be* $(1+0+1+1+0+1) = 4$.)

Toilet paper shortage (Network Flow)

Town authorities of *Algotown* are planning for an impending virus outbreak. They want to plan for panic buying and taking cues from some other towns, they know that one of the items that the town may run out of is toilet paper. They have asked your help to figure out whether the toilet paper demand of all n residents can be met. They provide you with the following information:

- There are n residents $r_1, ..., r_n$, m stores $s_1, ..., s_m$, and p toilet paper suppliers $x_1, ..., x_p$.
- The demand of each of the residents in terms of the number of rolls required.
- The list of stores that each of the residents can visit and purchase rolls from. A store cannot put any restriction on the number of rolls a customer can purchase given that those many rolls are available at the store.
- The number of rolls that supplier x_j can supply to store s_i for all $i \in \{1, ..., m\}$ and $j \in \{1, ..., p\}$.

The above information is provided in the following data structures:

- A 1-dimensional integer array $D[1...n]$ of size n, where $D[i]$ is the demand of resident r_i.
- A 2-dimensional 0/1 array $V[1...n, 1...m]$ of size $n \times m$, where $V[i, j] = 1$ if resident r_i can visit store s_j and $V[i, j] = 0$ otherwise.
- A 2-dimensional integer array $W[1...m, 1...p]$ of size $m \times p$, where $W[i, j]$ is the number of rolls of toilet paper that the supplier x_j can supply to store s_i.

Design an algorithm to determine if the demand of all residents can be met. That is, given (n, m, p, D, V, W) as input, your algorithm should output "yes" if it is possible for all residents to obtain the required number of rolls and "no" otherwise.

(*For example, consider that the town has two residents, one store and two suppliers. If* $D = [2, 2], V = [\begin{smallmatrix}1\\1\end{smallmatrix}]$, *and* $W = [2, 2]$, *then the demand can be met. However, if* $D = [2, 2], V = [\begin{smallmatrix}1\\1\end{smallmatrix}]$, *and* $W = [2, 1]$, *then the demand cannot be met.*)