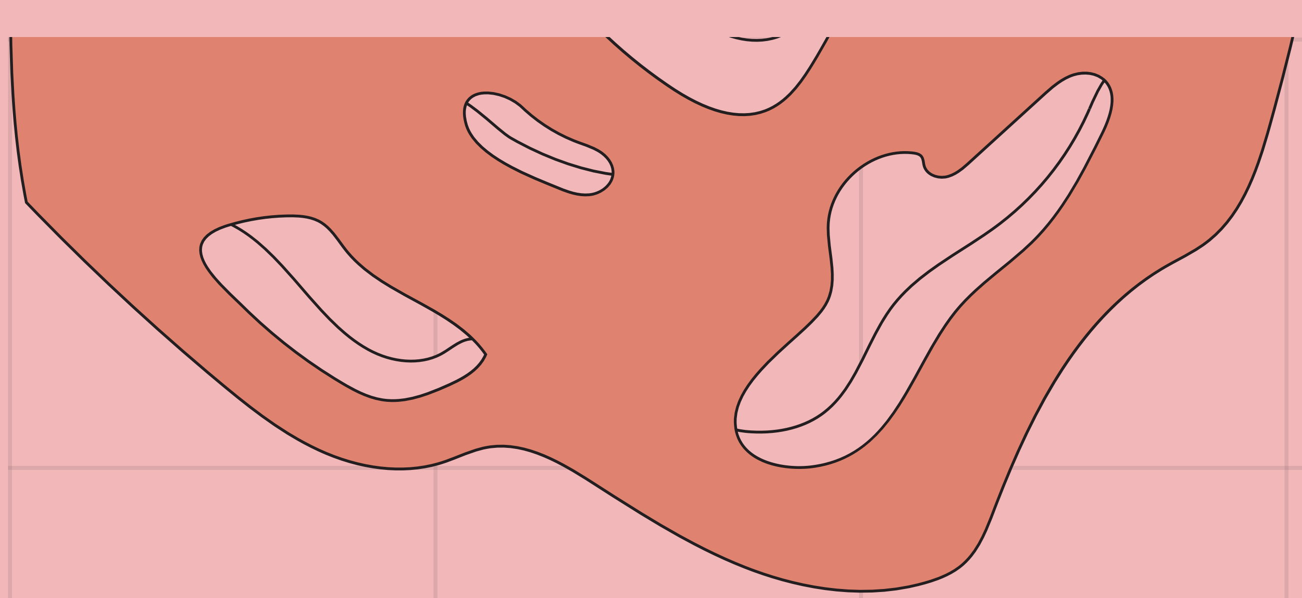


Team members:
210010054
210010019
210010032
210010017
210010023

Real life application of graph

1. Cheapest Flights within k stops

Problem statement:

In the actual world, cities are connected by flights. Either a direct trip or a flight with several stops between cities is an option for travelling between cities. Most people would rather save their money. When travelling by stopover and paying for more than one flight, we can sometimes save money because direct flights are generally more expensive. So, we are going to find the cheapest price to reach from current to destination city with limited stops.

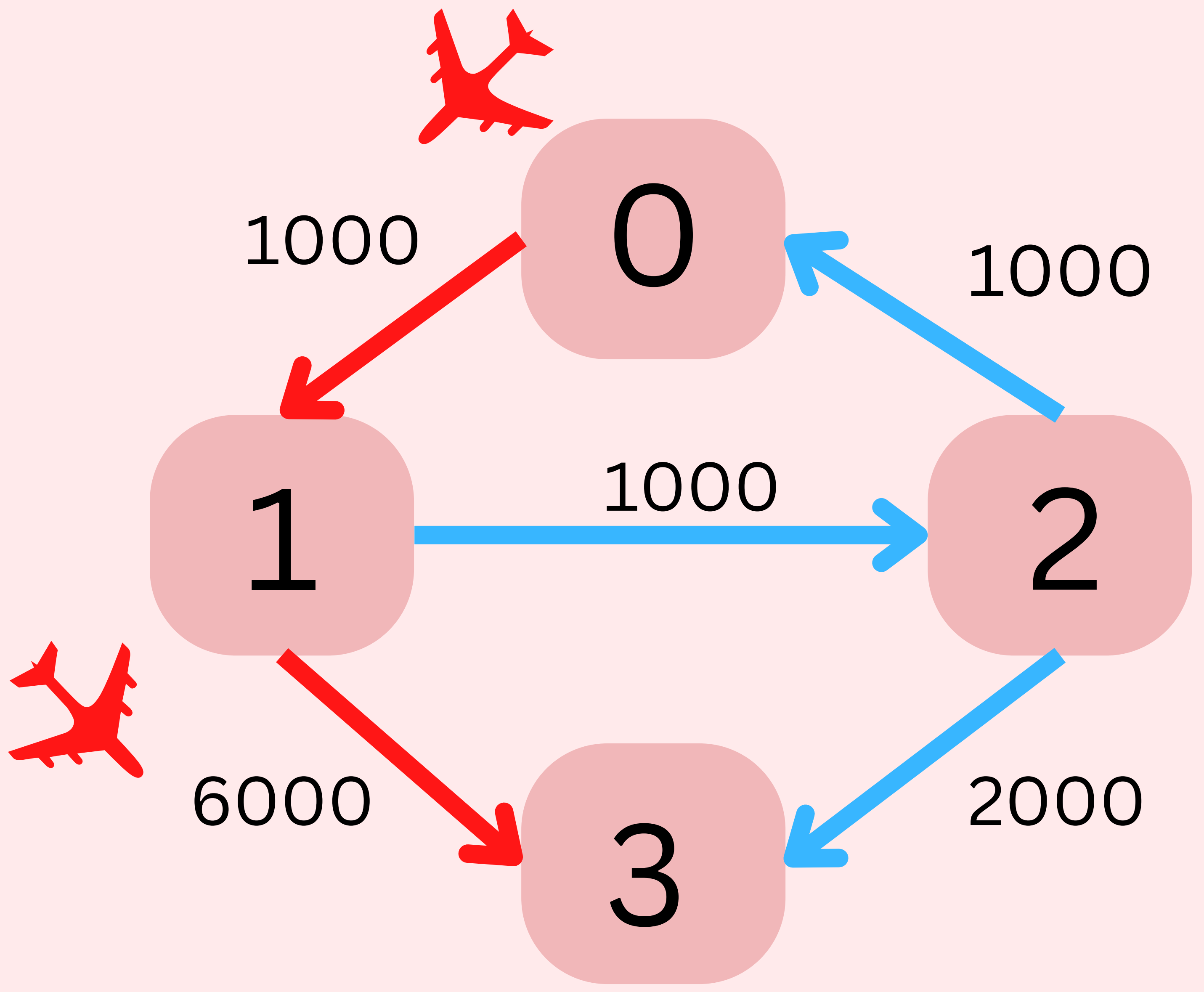
Given:

We have n cities connected by some number of flights. We are given an array of flights where flights[i] = [*from, to*, *price*] indicates that there is a flight from city "*from*" to city "*to*" with cost "*price*". We are representing city by non-negative integers.
Now, we are given three integers source, destination, and k, we will return the cheapest price from source to destination with at most k stops. If there is no such route, return -1.

Given: n = 4, flights = [[0,1,1000],[1,2,1000],[2,0,1000],[1,3,6000],[2,3,2000]], source = 0, destination = 3, k = 1

Answer: 7000

Explanation:
The graph is shown here.
The ideal route from city 0 to city 3 with atmost 1 stop possible is highlighted in red and costs 1000 + 6000 = 7000.
The route through the cities [0,1,2,3] is less expensive but incorrect because it requires 2 stops.

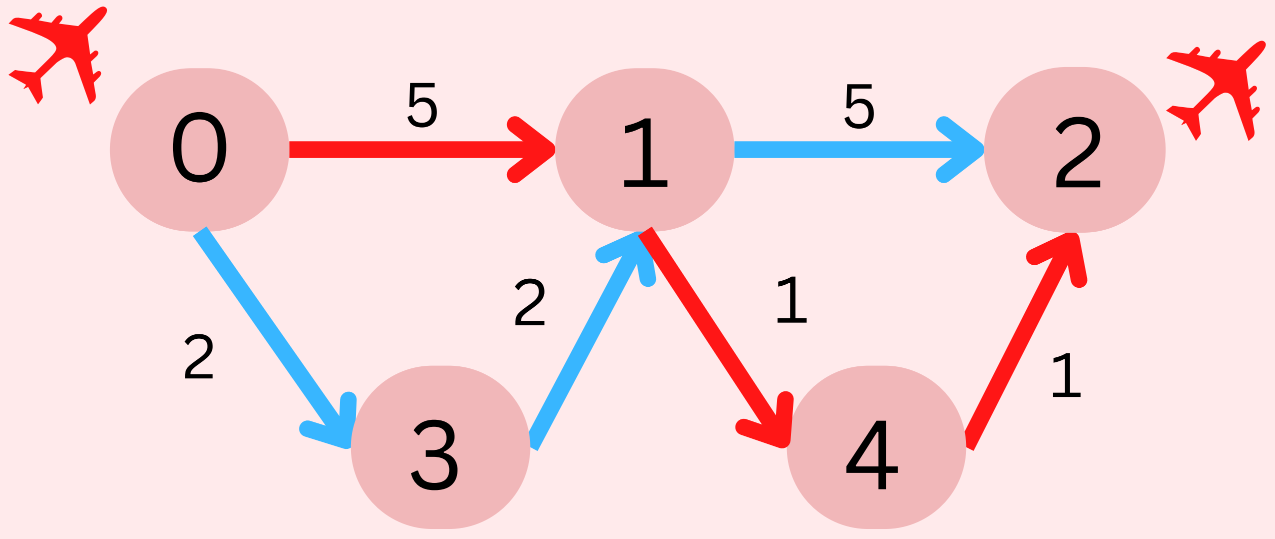


Solution:

Consider this graph.
source = 0, destination = 2 and k = 2

Step 1: Create graph (Adjacency List). Let's take a Queue which will store set of three values {stops, node, distance} and a cost array which will store cost of ith node. Now, we will initialize cost array with infinity.

Step 2: cost[source] = 0. Add {0, 0, 0} to Queue.



0 -> 1, 3
1 -> 2, 4
2 ->
3 -> 1
4 -> 2

{0, 0, 0}

{stops, node, distance}

Queue

cost[] =

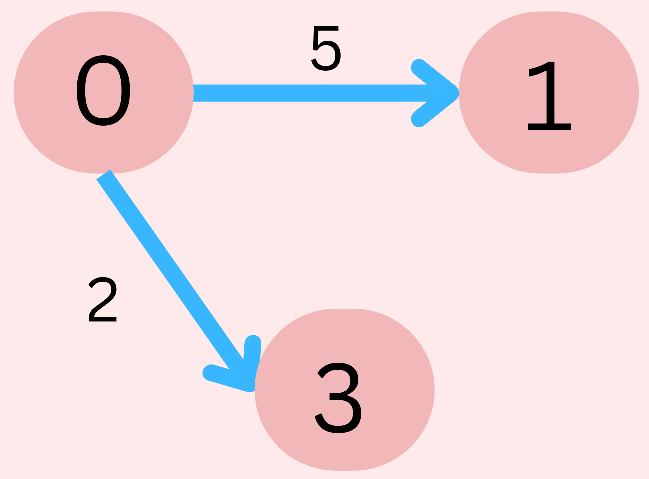
0	1	2	3	4
0	∞	∞	∞	∞

Step 3:

We will dequeue {0, 0, 0} initially. Since 0's neighbours are 1 and 3, the following condition will be examined:

cost of the present node + the cost of travelling to its neighbour < the neighbor's cost.

If this criterion is met, we will update the cost of the neighbour, which is calculated as follows: cost of the current node + the cost of travelling to its neighbour. Then we will increase stop value (here 0) by 1 and we will add updated {stops, node, cost} (in this case {1, 1, 5} and {1, 3, 2}) into the queue.



{1, 3, 2}
{1, 1, 5}
~~{0, 0, 0}~~

{stops, node, cost}

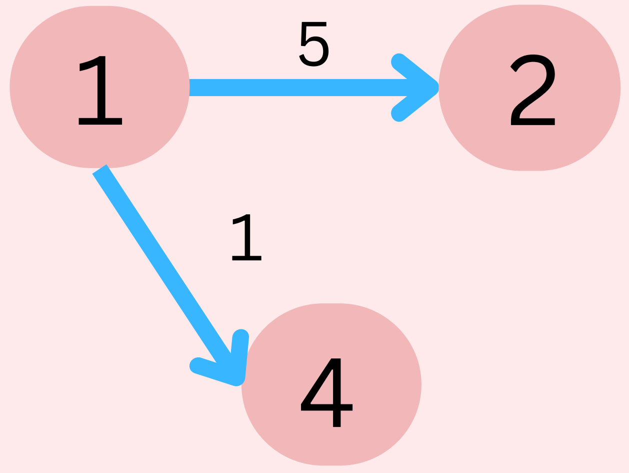
Queue

	0	1	2	3	4
cost[] =	0	5	∞	2	∞

Step 4:

We will dequeue {1, 1, 5}. 1's neighbours are 1 and 4,
since cost[1] + 5 < cost[2] i.e. 5 + 5 < ∞
=> cost[2] = 5 + 5 = 10
Then we will increase stop value (here 1) by 1 and we will add {2, 2, 10} into the queue.

cost[1] + 1 < cost[4] i.e. 5 + 1 < ∞
=> cost[4] = 5 + 1 = 6
Then we will increase stop value (here 1) by 1 and we will add {2, 4, 6} into the queue.



{2, 4, 6}
{2, 2, 10}
{1, 3, 2}
~~{1, 1, 5}~~
~~{0, 0, 0}~~

{stops, node, cost}

Queue

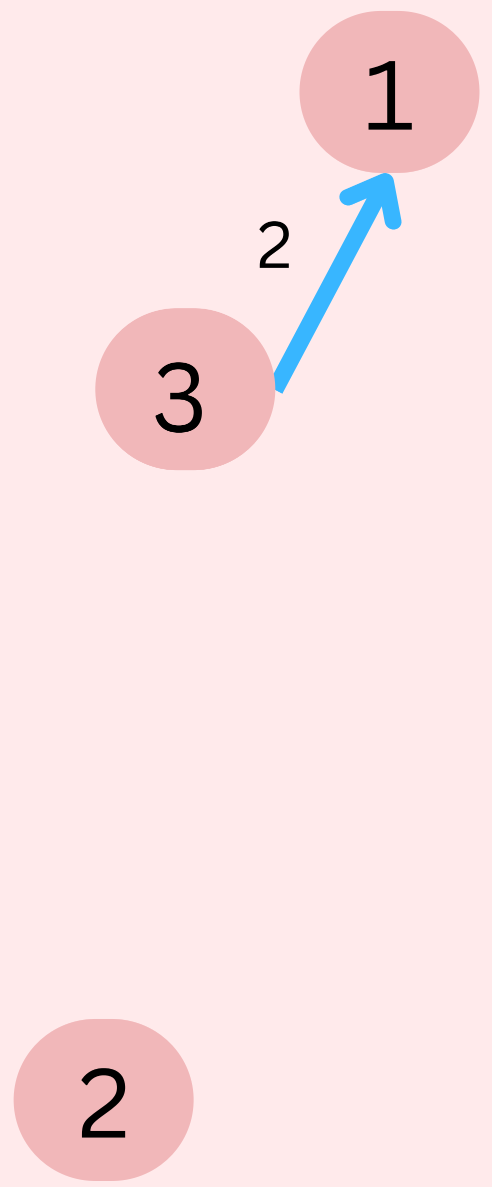
	0	1	2	3	4
cost[] =	0	5	10	2	6

Step 5:

We will dequeue {1, 3, 2}. 3's neighbour is 1, since cost[3] + 2 < cost[1] i.e. 2 + 2 < 5
=> cost[1] = 2 + 2 = 4
We will increase stop value (here 1) by 1 and we will add {2, 1, 4} into the queue.

Step 6:

Dequeue {2, 2, 10}. 2 doesn't have neighbour. So, we don't need to do any changes.



{2, 1, 4}
{2, 4, 6}
~~{2, 2, 10}~~
~~{1, 3, 2}~~
~~{1, 1, 5}~~
~~{0, 0, 0}~~

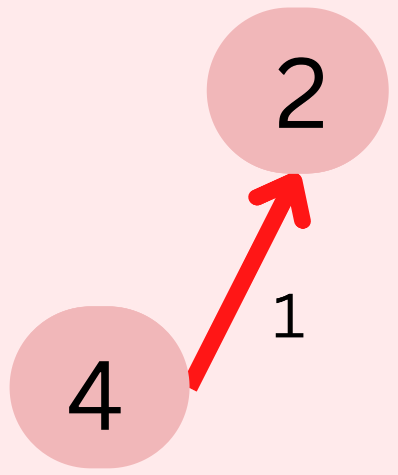
{stops, node, cost}

Queue

	0	1	2	3	4
cost[] =	0	4	10	2	6

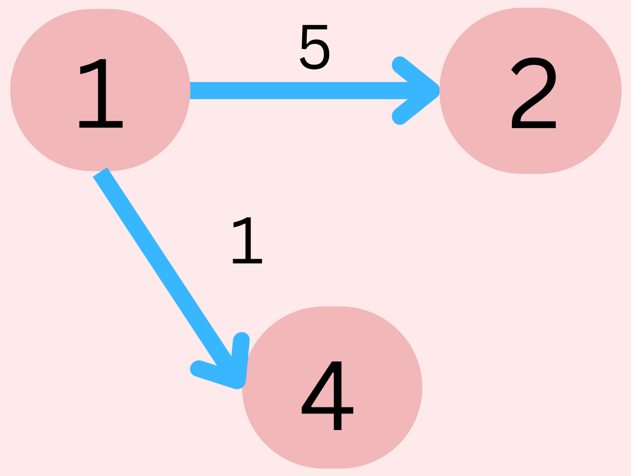
Step 7:

Dequeue {2, 4, 6}. 4's neighbour is 2,
cost[4] + 1 < cost[2] i.e. 6 + 1 < 10
=> cost[2] = 6 + 1 = 7   => stop = 2 + 1 = 3
Add {3, 2, 7} into the queue.



Step 8:

Dequeue {2, 1, 4}. 1's neighbours are 2 and 4.
cost[1] + 5 $\nless$ cost[2] i.e. 4 + 5 $\nless$ 7
So, we will not update cost[2].
cost[1] + 1 < cost[4] i.e. 4 + 1 < 6.
=> cost[4] = 4 + 1 = 5 => stop = 2 + 1 = 3;
Add {3, 4, 5} into the queue.



	0	1	2	3	4
cost[] =	0	4	7	2	5

{3, 4, 5}
{3, 2, 7}
~~{2, 1, 4}~~
~~{2, 4, 6}~~
~~{2, 2, 10}~~
~~{1, 3, 2}~~
~~{1, 1, 5}~~
~~{0, 0, 0}~~

{stops, node, cost}

Queue

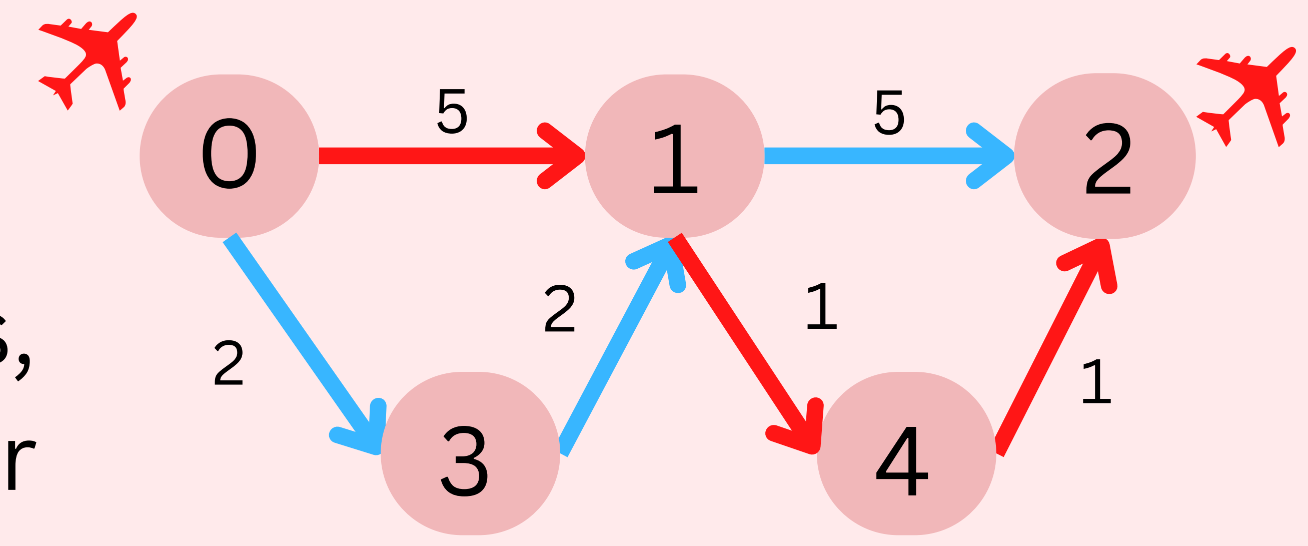
Step 9:

Dequeue {3, 2, 7}.

Now we have reached desired number of stops, even if we go further and found more cheaper alternative we have to increase stop by one, which we can't do since it is already at its maximum.

Thus , cost[2] = 7 which is our final answer.

Note:- If the cost[stop] = ∞ then there doesn't exist a path within k stops from source to destination.



{3, 4, 5}
~~{3, 2, 7}~~
~~{2, 1, 4}~~
~~{2, 4, 6}~~
~~{2, 2, 10}~~
~~{1, 3, 2}~~
~~{1, 1, 5}~~
~~{0, 0, 0}~~

{stops, node, cost}

Queue

cost[] =

0	1	2	3	4
0	4	7	2	5