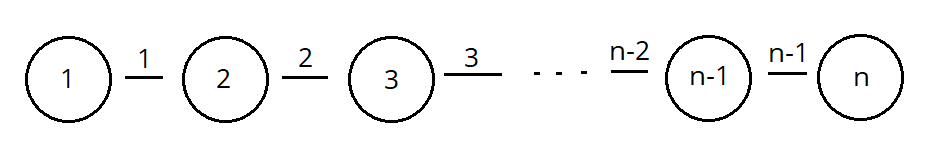
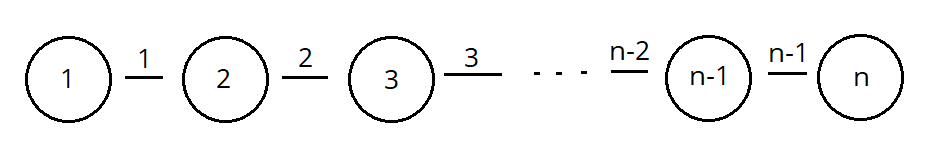
**Journey Tickets**

Prepared by Mihail Aleksiev

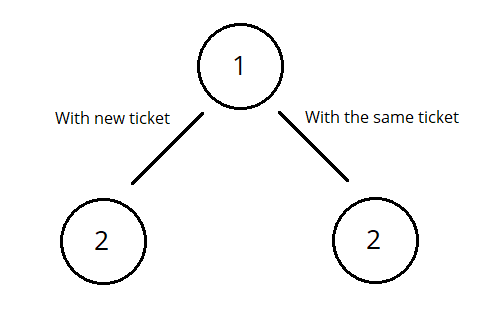
A "journey" is a sequence of flights : N (number of flights)

We have (n-1) stops on the airports, between First flight and Last flight.



For N = 1 (number of flights)

There is only one option, from airport A -> B (airport)

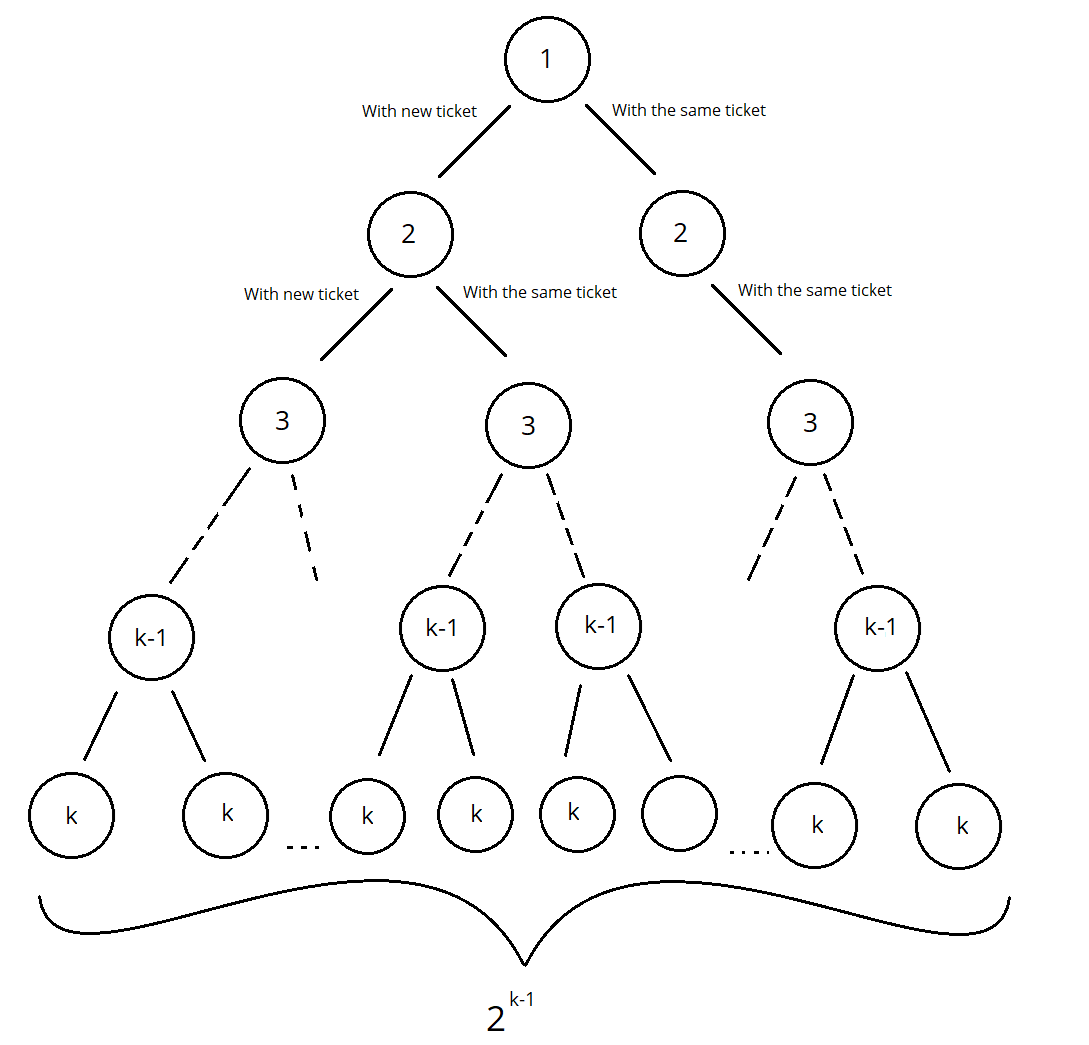


For N = 2 (number of flights)

After first flight, there is two options:

* breakup and continue with new ticket
* continue with the same ticket

For N = k (number of flights)



After every flight, which is not last, there is two options:

* breakup and continue with new ticket
* continue with the same ticket

This is binary tree with:

* root – first flight
* leaf – last flight

After every flight (which is not last), we have 2 choices

and we have (k-1) flights (after which we made choice).

Total number of possible routes are 2^(k-1)

Every route is different from the other.

//In my implementation I used binary tree.

struct Flight

{

int flight\_num; //Flight number

Flight\* breakup; //pointer to next flight

Flight\* without\_breakup; //pointer to next flight

Flight(int flight\_num)

{

this->flight\_num = flight\_num;

this->breakup = nullptr;

this->without\_breakup = nullptr;

}

};

bool IsLastFlight(Flight\* Flight) // Check if it is the last flight

{

return (Flight->breakup == nullptr && Flight->without\_breakup == nullptr);

}

int counter = 1; //Global variables for counting number of all Breakups.

Flight\* InsertJourney(int n, Flight\* First\_Flight, int i) // Recursive function which made a binary tree

{

if (i == 0) {} //end of the recursion

else

{

if (n < 0)

{

Flight\* temp = new Flight(n + i);

First\_Flight = temp;

First\_Flight->breakup = InsertJourney(n, First\_Flight->breakup, i - 1);

First\_Flight->without\_breakup = InsertJourney(-n, First\_Flight->without\_breakup, i - 1);

}

else

{

Flight\* temp = new Flight(n - i);

First\_Flight = temp;

First\_Flight->breakup = InsertJourney(-n, First\_Flight->breakup, i - 1);

First\_Flight->without\_breakup = InsertJourney(n, First\_Flight->without\_breakup, i - 1);

}

}

return First\_Flight;

}

//When there is breakup, the flight\_number will be negative (flight\_number < 0)

//This helps me when print all roads of the tree.

//If the next flight has negative flight number, means the ticket finished and need to use another ticket.

void MakeJourneyRoute(Flight\* Flight)

{

vector<int> route; //made array, where I will save routes.

PrintJourneyTickets(Flight, route);

}

void PrintJourneyTickets(Flight\* Flight, vector<int>& route)//Recursive function

{

if (Flight == nullptr) //end of recursion

{

return;

}

route.push\_back(Flight->flight\_num);

if (IsLastFlight(Flight)) //Executed when recursion stop

{

cout << "Breakup" << " " << counter << endl << ".";

for (int flight\_num : route)

{

if (flight\_num < 0)//if number of the flight is negative, the flight will be

{ // on another thicket

cout << endl << "." << -(flight\_num) << " "; //convert number to positive

}

else //if number of the flight is not negative, the flight will be on the same ticket

{

cout << flight\_num << " ";

}

}

cout << endl << endl;

counter++; //counting number of all breakups

}

PrintJourneyTickets(Flight->without\_breakup, route);

PrintJourneyTickets(Flight->breakup, route);

route.pop\_back();//delete the road, which is already printed

}

int main()

{

int N; //number of flights

cin >> N;

int unsigned i = N;

Flight\* First\_Flight = new Flight(0); //made first flight

MakeJourneyRoute(InsertJourney(N, First\_Flight, i));//function which print all roads form //root of the tree(first flight) to every leaf of the tree (every last flight)

return 0;

}