Template\_1

#include<iostream>

#include<stdio.h>

using namespace std;

int main() {

freopen("input/trail.txt", "r", stdin); // Reading from input file

freopen("output/trail.txt", "w", stdout); // Writing to output file

// Code

return 0;

}

Template\_2

#include<iostream>

#include<iomanip>

#include<stdio.h>

#include<string>

#include<string.h>

#include<math.h>

#include<algorithm>

#include<stdlib.h>

#include<unordered\_map>

using namespace std;

#define MIN(a,b) (((a)<(b))?(a):(b))

#define MAX(a,b) (((a)>(b))?(a):(b))

long long gcd(long long a, long long b) {

if(a==0) return b;

return gcd(b%a, a);

}

long long lcm(long long a, long long b) {

return ( (a/gcd(a,b)) \* b );

}

int main() {

freopen("input/trail.txt", "r", stdin); // Reading from input file

freopen("output/trail.txt", "w", stdout); // Writing to output file

// Code

return 0;

}

Observations:

* traffic light at the end of every street
* **at most one traffic light** will be green at each intersection at any given time
* When the light is green, one car can cross the intersection every second
* The schedule is a list of pairs: incoming street and duration
* **Each street can appear at most once** in the schedule
* schedule can ignore some of the incoming streets – those will never get a green light.
* By default all lights on all intersections are red
* Initially, all cars start at the end of the first street in their path, waiting for the green light (in case the trac light is red), or ready to move (if it's green)
* If two cars start at the end of the same street, the car listed first in the input goes first.
* **Scoring criteria** => F + (D – T) points if T ≤ D (if car reaches destination)

Else 0

Submission:

describes the traffic light schedules you want to set for specific intersections

* Apply minimum spanning tree [time associated with roads are corresponding edge weights].
* Objective: to find the path with minimum cost.
* Round robin scheduling for 2nd lvl optimisation.