TASK 3

- To optimize the path for the first drone, we need to calculate the shortest path and record the time at which it arrives at each point (i,j).

- Next, for the second drone, we need to check if there are any other drones occupying the same position at the current time. If there are, we should mark the corresponding box as blocked and find an alternate shortest path.

- We should repeat this process for all the remaining drones until we have found the optimal paths for all of them.

Code (C++)

```
#include<bits/stdc++.h>
using namespace std;

vector<int> dir{1,0,-1,0,1};    // used to traverse in 4 directions
vector<vector<int>> BFS(int i,int j,int x,int y,int t,map<pair<int,int>,unordered_set<int>> &m)
{
  vector<vector<int>>
path,dist(20,vector<int>(20,INT_MAX)),par(20,vector<int>(20,0)),time(20,vector<int>(20,0));
          // dist=> for storing min dist from a source here source is (i,j)
        // par=> par vector is used to find the shortest path from source to end
        // time=> time at which drone is present at (i,j)
  queue<vector<int>> q;   // queue used for BFS(source(i,j),time)
  q.push({i,j,t});
  dist[i][j]=-1;
  while(!q.empty())
  {
    auto f=q.front();
    q.pop();
    if(f[0]==x and f[1]==y)
    break;
    for(int i=0;i<4;i++)          // traversing at 4 directions
    {
      int a=f[0]+dir[i],b=f[1]+dir[i+1];
      if(a>=0 and b>=0 and a<20 and b<20 and dist[a][b]>dist[f[0]][f[1]]+1)   // checking if a,b is valid and
checking if it is visited or not
      {
        if(m.count({a,b})==0 or m[{a,b}].count(f[2]+1)==0)     // checking if any drone is present on (a,b) and
time current(f[2])+1
        {
          dist[a][b]=dist[f[0]][f[1]]+1;
          par[a][b]=(f[0]*20+f[1]);         // storing parent information
          q.push({a,b,f[2]+1});
          time[a][b]=f[2]+1;                // string time of current drone at (a,b)
        }

      }
    }
  }
```

```
    while(true)             // finding the shortest path using parent info
    {
        m[{x,y}].insert(time[x][y]);        // stroring the time for cur done at curr position
        if(x==i and y==j)                   // breaking the loop when we reached the starting posotion
        {
            path.push_back({x,y});
            break;
        }
        path.push_back({x,y});
        int temp=x;
        x=par[x][y]/20;                 // fetching x by diving with 20 as we stored in such fashion
        y=par[temp][y]%20;
    }
    reverse(path.begin(),path.end());   // reversing the vector as we started in reverse dir.
    return path;
}
int main()
{
    //Number of drones
    int n;
    cin>>n;
    //
    vector<vector<int>> v(n,vector<int>(5));
    map<pair<int,int>,unordered_set<int>> m;
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<5;j++)
        {
            cin>>v[i][j];
        }
        m[{v[i][0],v[i][1]}].insert(v[i][4]);       // time of entering of cur drone
    }
    for(int i=0;i<n;i++)
    {
        vector<vector<int>> path=BFS(v[i][0],v[i][1],v[i][2],v[i][3],v[i][4],m);
        for(int j=0;j<path.size();j++)
        {
            cout<<path[j][0]<<' '<<path[j][1];
            if(j+1<path.size())
            cout<<" -> ";
        }
        cout<<endl<<endl<<endl;
    }
    return 0;
}
```

INPUT

```
5               => Number of drones
0 0 19 19 6     => First 2 elements are starting coordinates than next two elements are destination and last one is time to start fly for drone 1
1 1 19 18 5     => Similar input for drone 2
7 8 15 16 4     => For drone 3
2 6 14 18 6     => For drone 4
7 6 13 16 5     => For done 5
```

OUTPUT

Path for drone 1

```
0 0 -> 1 0 -> 2 0 -> 3 0 -> 4 0 -> 5 0 -> 6 0 -> 7 0 -> 8 0 -> 9 0 -> 10 0 -> 11 0 -> 12 0 -> 13 0 -> 14 0 -> 15 0 ->
16 0 -> 17 0 -> 18 0 -> 19 0 -> 19 1 -> 19 2 -> 19 3 -> 19 4 -> 19 5 -> 19 6 -> 19 7 -> 19 8 -> 19 9 -> 19 10 -> 19
11 -> 19 12 -> 19 13 -> 19 14 -> 19 15 -> 19 16 -> 19 17 -> 19 18 -> 19 19
```

Path for Drone 2

```
1 1 -> 2 1 -> 3 1 -> 4 1 -> 5 1 -> 6 1 -> 7 1 -> 8 1 -> 9 1 -> 10 1 -> 11 1 -> 12 1 -> 13 1 -> 14 1 -> 15 1 -> 16 1 ->
17 1 -> 18 1 -> 19 1 -> 19 2 -> 19 3 -> 19 4 -> 19 5 -> 19 6 -> 19 7 -> 19 8 -> 19 9 -> 19 10 -> 19 11 -> 19 12 ->
19 13 -> 19 14 -> 19 15 -> 19 16 -> 19 17 -> 19 18
```

Path for Drone 3

```
7 8 -> 8 8 -> 9 8 -> 10 8 -> 11 8 -> 12 8 -> 13 8 -> 14 8 -> 15 8 -> 15 9 -> 15 10 -> 15 11 -> 15 12 -> 15 13 -> 15
14 -> 15 15 -> 15 16
```

Path for Drone 4

```
2 6 -> 3 6 -> 4 6 -> 5 6 -> 6 6 -> 7 6 -> 8 6 -> 9 6 -> 10 6 -> 11 6 -> 12 6 -> 13 6 -> 14 6 -> 14 7 -> 14 8 -> 14 9 ->
14 10 -> 14 11 -> 14 12 -> 14 13 -> 14 14 -> 14 15 -> 14 16 -> 14 17 -> 14 18
```

Path for Drone 5

```
7 6 -> 8 6 -> 9 6 -> 10 6 -> 11 6 -> 12 6 -> 13 6 -> 13 7 -> 13 8 -> 13 9 -> 13 10 -> 13 11 -> 13 12 -> 13 13 -> 13
14 -> 13 15 -> 13 16
```