Guia de datos

Al ejecutar las simulaciones se generan datos que su algoritmo va recibir (definen el escenario apra este evento o simulacion de ciduad), datos de la simulacion y resultados. En esta guia vamos a repasar cada uno de ellos.

Datos de Input

La estructura de codigo de los inputs lo pueden ver al inicio de la funcion de `plan_evacuation` en el `solution.py`.

```python
def plan_evacuation(self, city: CityGraph, proxy_data: ProxyData,
                    max_resources: int) -> PolicyResult:
    """
    Plan the evacuation route and resource allocation.

    Args:
        city: The city layout with nodes and edges
            - city.graph: NetworkX graph with the city layout
            - city.starting_node: Your starting position
            - city.extraction_nodes: List of possible extraction points

        proxy_data: Information about the environment
            - proxy_data.node_data[node_id]: Dict with node indicators
            - proxy_data.edge_data[(node1,node2)]: Dict with edge indicators

        max_resources: Maximum total resources you can allocate

    Returns:
        PolicyResult with:
        - path: List[int] - List of node IDs forming your evacuation path
        - resources: Dict[str, int] - How many of each resource to take:
                    {'explosives': x, 'ammo': y, 'radiation_suits': z}
                    where x + y + z <= max_resources
    """
    print(f'City graph: {city.graph} \n')
    print(f'City starting_node: {city.starting_node}\n')
    print(f'City extraction_nodes: {city.extraction_nodes}\n')
    print(f'Proxy node_data: {proxy_data.node_data} \n \n')
    print(f'Proxy edge_data: {proxy_data.edge_data} \n \n')
    print(f'Max Resources: {max_resources} \n \n')
```

PROF

Su algoritmo en `public/tools/simulator.py` recibira 3 datos (variables input):

1. `city [Objeto con networkx y otros valores]`, representa lo que conocemos de la ciudad. Conexiones entre nodos de la ciudad, nodo inicial, y nodos de extraccion.
   1.1 `city.graph [nx.Graph]`: Es un archivo de tipo `networkx` que es la libreria de grafos mas usada en python. Video Introductorio a NetworkX. Representa las conecciones **"esperadas"** de la ciudad.La ventaja de utilizar este objeto es que pueden aplicar algoritmos, metodos o lo que deseen the la libreria `networkx` sin necesidad de hacer el codigo manualmente, esto incluye visualizaciones y otras cosas interesantes. Esto se ve reflejado en el algoritmo `Naive` que se proporciona de inicio, ya que usamos el metodo de esta libreria para calcular el **shrotest** path de manera automatica sobre este objeto. Mi recomendacion es que utilicen esta estructura y esta libreria para ejecutar sus algoritmos, sin embargo pueden convertirla a una **linked_list** y usar las cosas de manera manual. Para entender mejor que puedes hacer con este objecto lee la documentacion 'docs/b_tutorial_nx.md'.

```
    path = nx.shortest_path(city.graph, city.starting_node, target,
                            weight='weight')
```

1.2 `city.starting_node [Dict]`: es un entero que representa el nodo donde empezaran su equipo al que tienen que extraer.

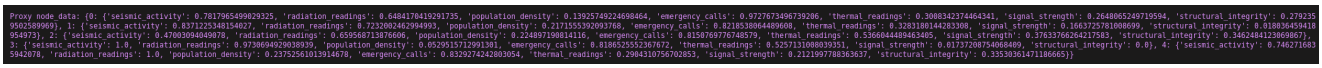
1.3 `city.extraction_nodes [Dict]`: es una lista con los nodos que son de extraccion. tienen que llegar a cualquiera de los dos para que la extraccion se de como valida.

```
City graph: Graph with 5 nodes and 9 edges

City starting_node: 4

City extraction_nodes: [0, 2]
```

2. `proxy_data [Objeto de Diccionarios]`: es un diccionario (json) que contiene los proxies o datos que reflejan las condiciones o el estado de la ciudad o grafica.

   2.1 `proxy_data.node_data [Dict]`: So los datos relacionados al **nodo**, la llave es el nodo, y el diccionario las variables y los valores para cada nodo. `{'nodo': {'caracteristica':'valor'}}`. Son las caracteristicas que conocemos del lugar.

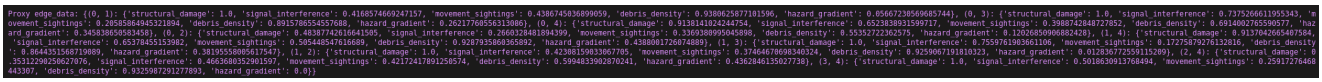
```
proxy_data.node_data ={
    '0':
        {'caracteristica_1':valor, ..., 'caracteristica_n':valor},
    ...,
    `n`:
        {'caracteristica_1':valor, ..., 'caracteristica_n':valor}
}
```



2.2 `proxy_data.edge_data [Dict]]`: Son los datos relacionados al **edge**, la llave es una tupla que representa la conexion entre dos nodos **(nodo1, nodo2)**, y el diccionario las variables y los valores para cada nodo. `{'(nodo1, nodo2)':{'caracteristica':'valor'}}`. Son las caracteristicas que conocemos de las calles o conecciones entre los nodos.

PROF

```
proxy_data.edge_data ={
    '(0,20)':
        {'caracteristica_1':valor, ..., 'caracteristica_n':valor},
    ...,
    `(n,m)`:
        {'caracteristica_1':valor, ..., 'caracteristica_n':valor}
}
```



3. `max_resources [int]`: Es el numero maximo de recursos que pueden cargar las personas que

seran extraidas en esta ciudad. Puede variar, no es fijo.

```
Max Resources: 4
```

Dataframe

Adicionalmente se te proporciona una funcion que dado los inputs en JSON los convierte a un DataFrame de pandas por si deseas usarlo asi en vez de JSON. La funcion se importa en`public/student_code/solution.py` donde puedes decidir usarla o no. Los prints estaran comentados.

```
proxy_data_nodes_df = convert_node_data_to_df(proxy_data.node_data)
proxy_data_edges_df = convert_edge_data_to_df(proxy_data.edge_data)

print(f'\n Node Data: \n {proxy_data_nodes_df}')
print(f'\n Edge Data: \n {proxy_data_edges_df}')
```

```
 Edge Data:
    node_1  node_2  structural_damage  signal_interference  movement_sightings  debris_density  hazard_gradient
0       0       4           0.979880             0.684057            0.351825        0.949944         0.268663
1       0       3           0.909423             0.381737            0.288192        0.950377         0.000000
2       0       1           0.912706             0.436131            0.344049        0.897784         0.094216
3       0       2           1.000000             0.642489            0.188633        0.984381         0.357875
4       1       3           0.820032             0.488000            0.115081        0.712248         0.000000
5       1       2           1.000000             0.815363            0.027234        0.948156         0.286211
6       1       4           0.482526             0.614296            0.142075        0.635828         0.372275
7       2       3           0.930568             0.792695            0.100310        1.000000         0.356552
8       3       4           0.300114             0.505024            0.104159        0.693858         0.426225

 Node Data:
    node  seismic_activity  radiation_readings  population_density  emergency_calls  thermal_readings  signal_strength  structural_integrity
0     0          0.781797            0.648417            0.139257         0.972767          0.300834         0.264807              0.279236
1     1          0.837123            0.723200            0.217156         0.821854          0.328318         0.166373              0.018036
2     2          0.470031            0.659569            0.224897         0.815077          0.536604         0.376338              0.346248
3     3          1.000000            0.973069            0.052952         0.818653          0.525713         0.017372              0.000000
4     4          0.746272            1.000000            0.237526         0.832927          0.290431         0.212200              0.335304
```

Datos Output de tu algoritmo

Los datos que debe regresar tu algoritmo deben ser 2 estrucutras:

1. Una lista indicando la suscesion de nodos valida. Por ejemplo: `[5,7,8,2,34,56]` donde el nodo inicial es `5` (debe ser el mismo que indica el problema), un nodo de extraccion es `56`, y todo nodo sucesivo esta conectado osea de puede pasar al siguiente en la lista por que existe un camino valido. Si no es valido el nodo inicial, final o alguna conexion tu equipo perdera automaticamente. Puede ocurrir que haya un path valido, pero no tengas los recursos adecuados apra solucionar el evento que pase en el nodo, en cuyo caso tambien perderas.
2. Un diccionario con un valor valido apra cada recurso (en total tienen que ser menores o iguales al `max_resource`), si es mayor al permitido tu equipo se perdera automaticamente.
   2.1 Estos recursos deben de servirte para solucionar los problemas o eventos que encontraras por los nodos.