

PRUEBA

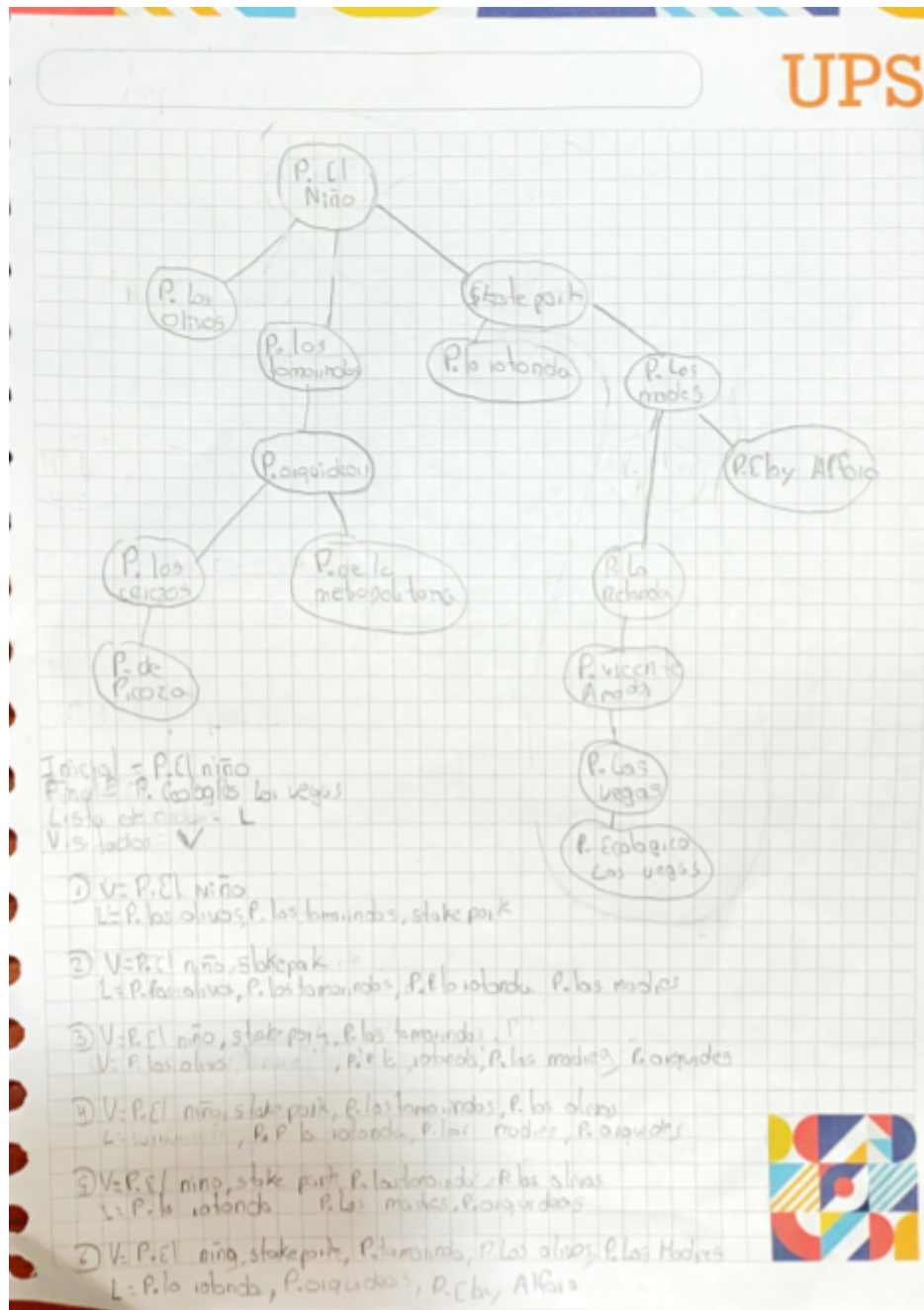
- Agregar un grafico con los nodos conformados.



-
- Hand-drawn phylogenetic tree on graph paper showing relationships between various populations of *P. el niño*. The tree is rooted at the top with the node *P. el niño*. The branches and nodes are labeled as follows:
- Root: *P. el niño*
 - Branch 1 (Left): *P. los Olivos* (997 m)
 - Branch 2 (Left): *P. los Llanos* (1000 m)
 - Branch 3 (Left): *P. Oquendo* (900 m)
 - Branch 4 (Left): *P. los Cejeños* (1600 m)
 - Branch 5 (Left): *P. Pácoro* (1100 m)
 - Branch 6 (Left): *P. La Mesa* (1220 m)
 - Branch 7 (Left): *P. Vicente Amador* (405 m)
 - Branch 8 (Left): *P. los Vagos* (300 m)
 - Branch 9 (Left): *P. Los Robles* (259 m)
 - Branch 10 (Right): *P. la Honda* (529 m)
 - Branch 11 (Right): *P. El Barrio* (404 m)
- Annotations on the left side of the tree:
- Inicio = *P. el niño*
 - Fin = *P. Ecología los Vagos*

- Realizar la búsqueda por amplitud, profunda y costo.

AMPLITUD



UPS

V: P. El niño, stake park, P. los olivos, P. las madres, P. la infancia
L: P. los quiches, P. Clay Alvaro, P. de la metropolitana, P. de la cecza

V: P. los quiches, P. Clay Alvaro
L: P. Clay Alvaro, P. de la metropolitana, P. de la cecza

V: P. los quiches, P. Clay Alvaro
L: P. de la metropolitana, P. de la cecza, la retonda

V: P. los quiches, P. Clay Alvaro, P. de la metropolitana
L: P. de la cecza, la retonda

V: P. los quiches, P. Clay Alvaro, P. de la metropolitana, P. de la cecza
L: la retonda, P. cecza

V: P. los quiches, P. Clay Alvaro, P. de la metropolitana, P. de la cecza, la retonda
L: P. cecza, P. las vegas

V: P. los quiches, P. Clay Alvaro, P. de la metropolitana, P. de la cecza, la retonda, P. cecza
L: P. las vegas, Parque Vicente

V: P. + P. las vegas
L: Parque Vicente, P. Ecologica las vegas

V: A. P. las vegas, Parque Vicente
L: P. Ecologica las vegas





PROFUNDIDAD

Amplitud

UPS

cc: 0

① V: P. El niño
L: P. Los olivos, P. los tamarindos, stakepark

cc: 101997
V: P. El niño, P. Los olivos

② cc: 0 + 1000
V: P. El niño, P. los tamarindos
L: P. Orquídeas

cc: 1000 + 500
V: P. El niño, P. los tamarindos
L: P. los cerezos, P. la metropolitana

cc: 1500 + 1100
V: P. El niño, P. los tamarindos, P. los cerezos,
L: P. Picoazo, P. la metropolitana

cc: 1600 + 1600
V: P. El niño, P. los tamarindos, P. los cerezos, P. Picoazo

cc: 1600 + 1120
V: P. El niño, P. los tamarindos, P. los metropolitana
L:

③ cc: 997 + 1000 + 1500
V: P. El niño, P. los tamarindos, stakepark
L: P. Elay Alfaro, P. la reboda

cc: 3497 + 1404
V: P. El niño, P. los tamarindos, stakepark, P. Elay Alfaro
L:


④ cc: 3901 + 529
V: P. El niño, P. los tamarindos, stakepark, P. la reboda
L: P. Vicente Amador

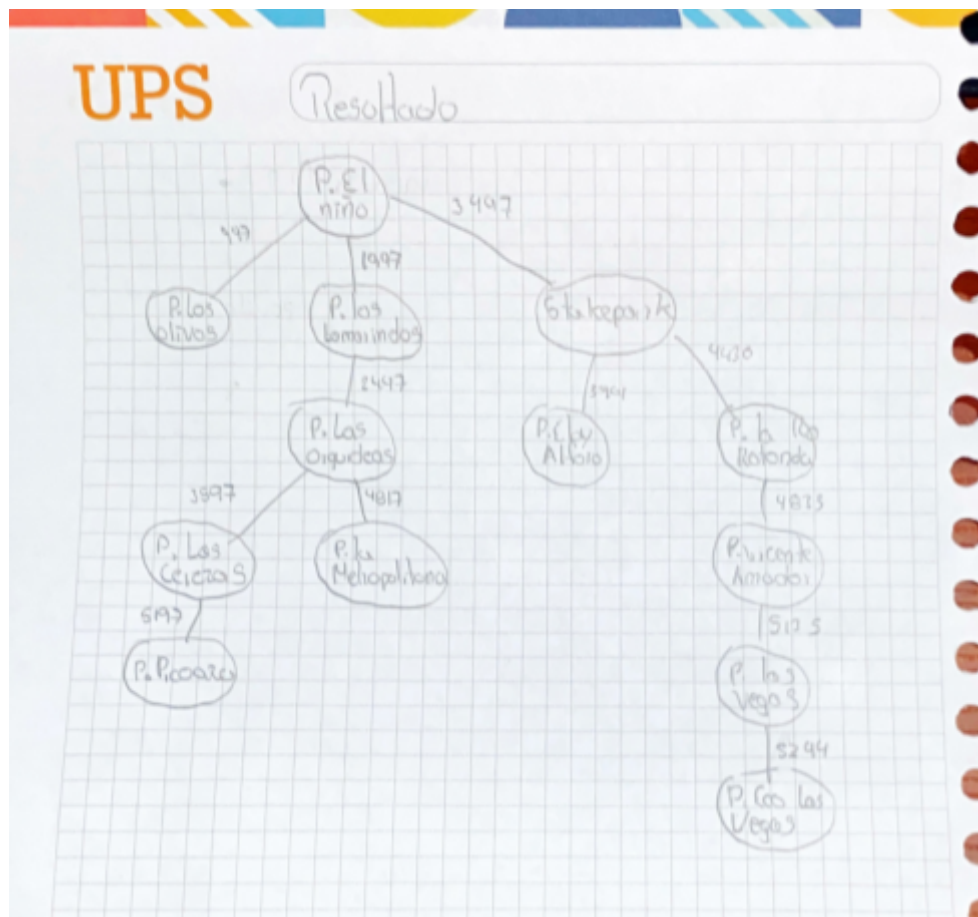
cc: 4430 + 409
V: P. El niño, P. los tamarindos, stakepark, P. la reboda, P. Vicente Amador
L: P. Los Vegas

cc: 4835 + 300
V: P. El niño, P. los tamarindos, stakepark, P. la reboda, P. Vicente Amador
L: P. Los Vegas

cc: 51351759

Resultado: V: P. El niño, P. los olivos, P. tamarindos, stakepark, P. Elay Alfaro, P. la reboda, P. Vicente Amador, P. los Vegas, P. Picoazo, P. los cerezos, P. la metropolitana





- Programar y presentar los resultados mediante los algoritmos de búsqueda.

AMPLITUD

In [6]:

```
import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import Image, display

Grafo=nx.DiGraph()

def view_pydot(pdot):
    plt = Image(pdot.create_png())
    display(plt)

class Node:
    def __init__(self, data, child=None): # Constructor de la clase
        self.data = data
        self.child = None
        self.fathr = None
        self.cost = None
        self.set_child(child)

    def set_child(self, child): # Agregar hijos
        self.child = child
        if self.child is not None:
            for ch in self.child:
                ch.fathr = self

    def equal(self, node): # Igual al equals de Java
        if self.data == node.data:
            return True
        else:
            return False

    def on_list(self, node_list): # Verificar si el nodo esta en la lista
        listed = False
        for n in node_list:
            if self.equal(n):
                listed = True
        return listed

    def __str__(self): # Igual al toString Java
        return str(self.data)

def graficar(datos):
    graf = nx.DiGraph()
    graf.add_nodes_from(datos)
    for valor, listaValor in datos.items():
        for a in listaValor:
            graf.add_edge(valor,a,size=250)

    plt.figure(figsize=(20,20))
    nx.draw_networkx(graf, node_color = 'yellow', with_label = True, node_size=2000)
    plt.show()

def graficarRes(grafo):
    p=nx.drawing.nx_pydot.to_pydot(grafo)
```

```
view_pydot(p)
```

```
# Implementacion del metodo de busqueda por amplitud
def search_Amplitud_solution2(connections, init_state, solution):
    solved = False # Variable para almacenar el estado de la busqueda
    visited_nodes = [] # Nodos visitados
    frontrs_nodes = [] # Nodos en busqueda o lista nodos

    init_node = Node(init_state) # Nodo inicial
    frontrs_nodes.append(init_node)
    while (not solved) and len(frontrs_nodes) != 0:
        node = frontrs_nodes[0]
        # extraer nodo y añadirlo a visitados
        visited_nodes.append(frontrs_nodes.pop(0))
        if node.data == solution: # Preguntar se el nodo obtenido es la solucion
            solved = True
            Grafo.add_node(node.data,color='red')
            return node # Retornamos el nodo de la solucion
        else:
            # expandir nodos hijo - ciudades con conexion
            node_data = node.data
            child_list = []
            for chld in connections[node_data]:
                child = Node(chld)
                child_list.append(child)
                if not child.on_list(visited_nodes) and not child.on_list(frontrs_nodes):
                    frontrs_nodes.append(child)
                    Grafo.add_edge(node.data,child)
                    if child.on_list(visited_nodes):
                        Grafo.add_node(node.data,color='red')
            node.set_child(child_list)

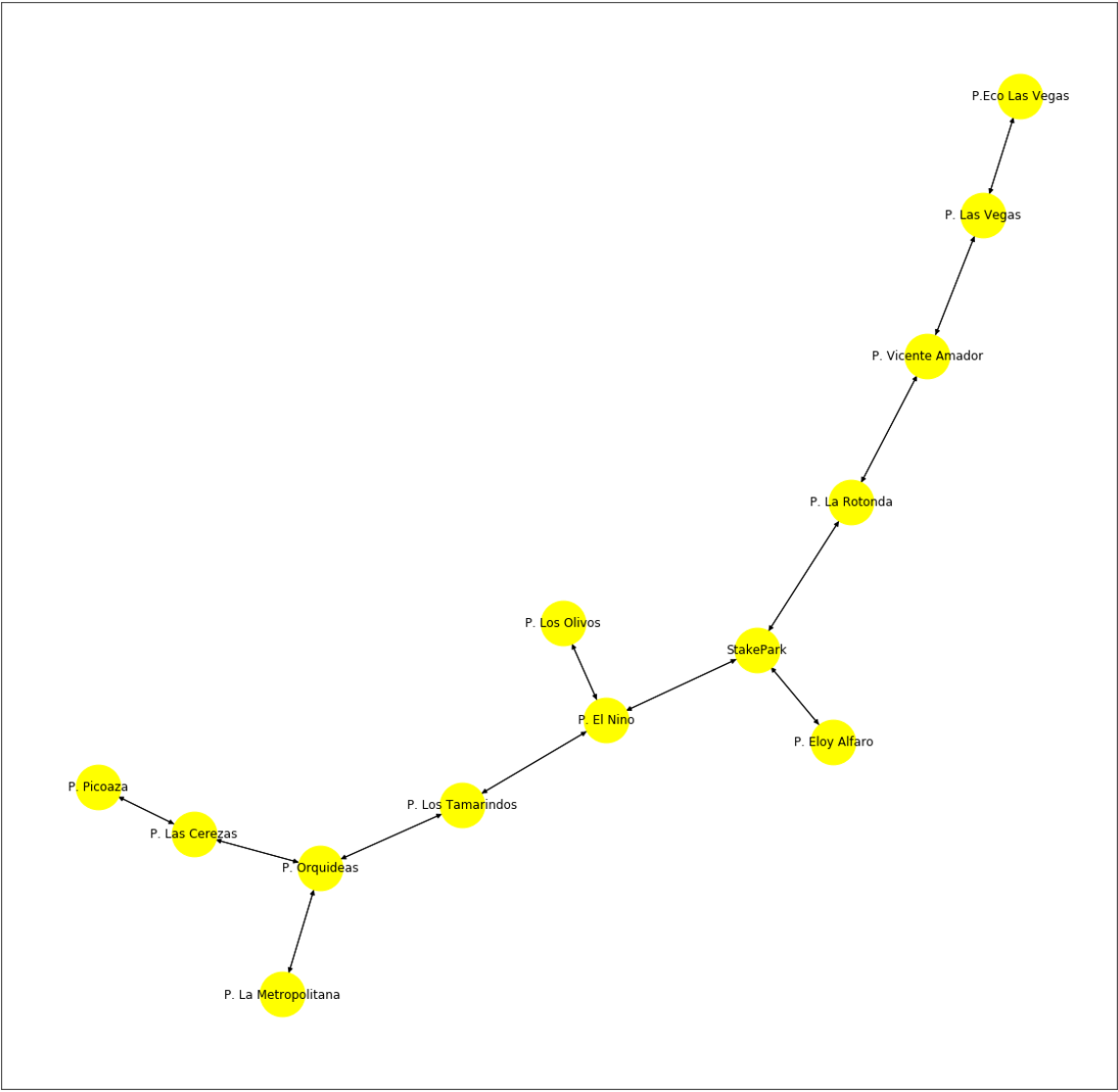
hospitales = {
    'P. El Nino': {'P. Los Olivos', 'P. Los Tamarindos', 'StakePark'},
    'P. Los Olivos': {'P. El Nino'},
    'P. Los Tamarindos': {'P. Orquideas', 'P. El Nino'},
    'P. Orquideas': {'P. Las Cerezas', 'P. La Metropolitana', 'P. Los Tamarindos'},
    'P. Las Cerezas': {'P. Picoaza', 'P. Orquideas'},
    'P. Picoaza': {'P. Las Cerezas'},
    'P. La Metropolitana': {'P. Orquideas'},
    'StakePark': {'P. La Rotonda', 'P. Eloy Alfaro', 'P. El Nino'},
    'P. Eloy Alfaro': {'StakePark'},
    'P. La Rotonda': {'P. Vicente Amador', 'StakePark'},
    'P. Vicente Amador': {'P. Las Vegas', 'P. La Rotonda'},
    'P. Las Vegas': {'P. Eco Las Vegas', 'P. Vicente Amador'},
    'P. Eco Las Vegas': {'P. Las Vegas'}
}

graficar(hospitales)

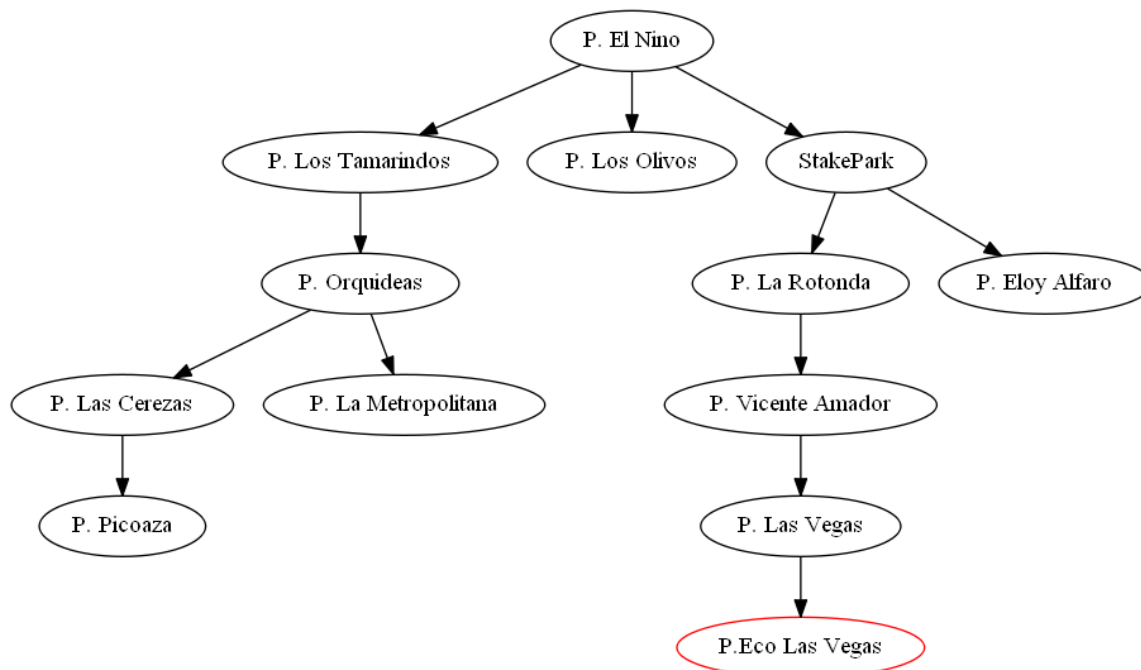
init_state = 'P. El Nino'
solution = 'P. Eco Las Vegas'
solution_node = search_Amplitud_solution2(hospitales, init_state, solution)
# mostrar resultado
result = []
node = solution_node
if node is not None:
```

```
while node.fathr is not None:
    result.append(node.data)
    node = node.fathr
result.append(init_state)
result.reverse() # Reverso el resultado (Solo para presentar)
print(result)
else:
    print("No hay solucion !!!!")

graficarRes(Grafo)
```



['P. El Nino', 'StakePark', 'P. La Rotonda', 'P. Vicente Amador', 'P. Las Vegas', 'P.Eco Las Vegas']



COSTOS

In [2]:

```

import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import Image, display

Grafo=nx.DiGraph()

def view_pydot(pdot):
    plt = Image(pdot.create_png())
    display(plt)

def graficar(datos):
    graf = nx.DiGraph()
    graf.add_nodes_from(datos)
    for valor, listaValor in datos.items():
        for a in listaValor:
            graf.add_edge(valor,a,size=250,length=str(listaValor[a]))

    pos = nx.spring_layout(graf)
    plt.figure(figsize=(20,20))
    #print(" ")
    labels = nx.get_edge_attributes(graf, 'length')
    #print(labels)
    nx.draw_networkx(graf, pos, node_color = 'green', with_labels = True, node_size=300
0)
    nx.draw_networkx_edge_labels(graf,pos,edge_labels=labels,font_color='black',font_si
ze=10)
    plt.show()

def graficarNodoResultado(grafo):
    p=nx.drawing.nx_pydot.to_pydot(grafo)
    for i, edge in enumerate(p.get_edges()):
        edge.set_label(str(edge.get_label()))
    view_pydot(p)

class Node:
    def __init__(self, data, child=None): # Constructor de la clase
        self.data = data
        self.child = None
        self.fathr = None
        self.cost = None # Importante tener el costo de recorrer el nodo
        self.set_child(child)

    def set_child(self, child): # Agregar hijos
        self.child = child
        if self.child is not None:
            for ch in self.child:
                ch.fathr = self

    def equal(self, node):
        if self.data == node.data:
            return True
        else:
            return False

    def on_list(self, node_list): # Verfiicar su el nodo esta en la lista
        listed = False
        for n in node_list:

```

```

        if self.equal(n):
            listed = True
        return listed

def __str__(self): # Igual al toString Java
    return str(self.data)

def Compare(node):
    return node.cost

# Implementacion del metodo de busqueda por costo
def search_costo_solucion(connections, init_state, solution,g):
    var=""
    solved = False # Variable para almacenar el estado de la busqueda
    visited_nodes = [] # Nodos visitados
    frontier_nodes = [] # Nodos en busqueda o lista nodos o nodos por visitar

    init_node = Node(init_state) # Nodo inicial
    init_node.cost = 0 # Agregar costo inicial
    frontier_nodes.append(init_node)
    while (not solved) and len(frontier_nodes) != 0:
        frontier_nodes = sorted(frontier_nodes, key=Compare) # Ordenar lista de nodos
        node = frontier_nodes[0]
        visited_nodes.append(frontier_nodes.pop(0)) # Extraer nodo y añadirlo a visitad
os
        if node.data == solution:# Solucion encontrada
            solved = True
            g.add_node(node.data,color='red')
            return node
        else:
            node_data = node.data# Expandir nodos hijo (ciudades con conexion)
            child_list = []
            for achild in connections[node_data]: # Recorrera cada uno de los nodos hij
os
                child = Node(achild)
                cost = connections[node_data][achild] # Obtener el costo del nodo
                child.cost = node.cost + cost # Agregamos el costo actual del nodo + el
historial
                child_list.append(child)
                if not child.on_list(visited_nodes):
                    if child.on_list(frontier_nodes): # Si está en la lista lo sustituim
os con el nuevo valor de coste si es menor
                        g.add_edge(node.data,child,label=child.cost)
                        for n in frontier_nodes:
                            if n.equal(child) and n.cost > child.cost:
                                frontier_nodes.remove(n)
                                frontier_nodes.append(child)
                        else:
                            g.add_edge(node.data,child,label=child.cost)
                            frontier_nodes.append(child)
                node.set_child(child_list)

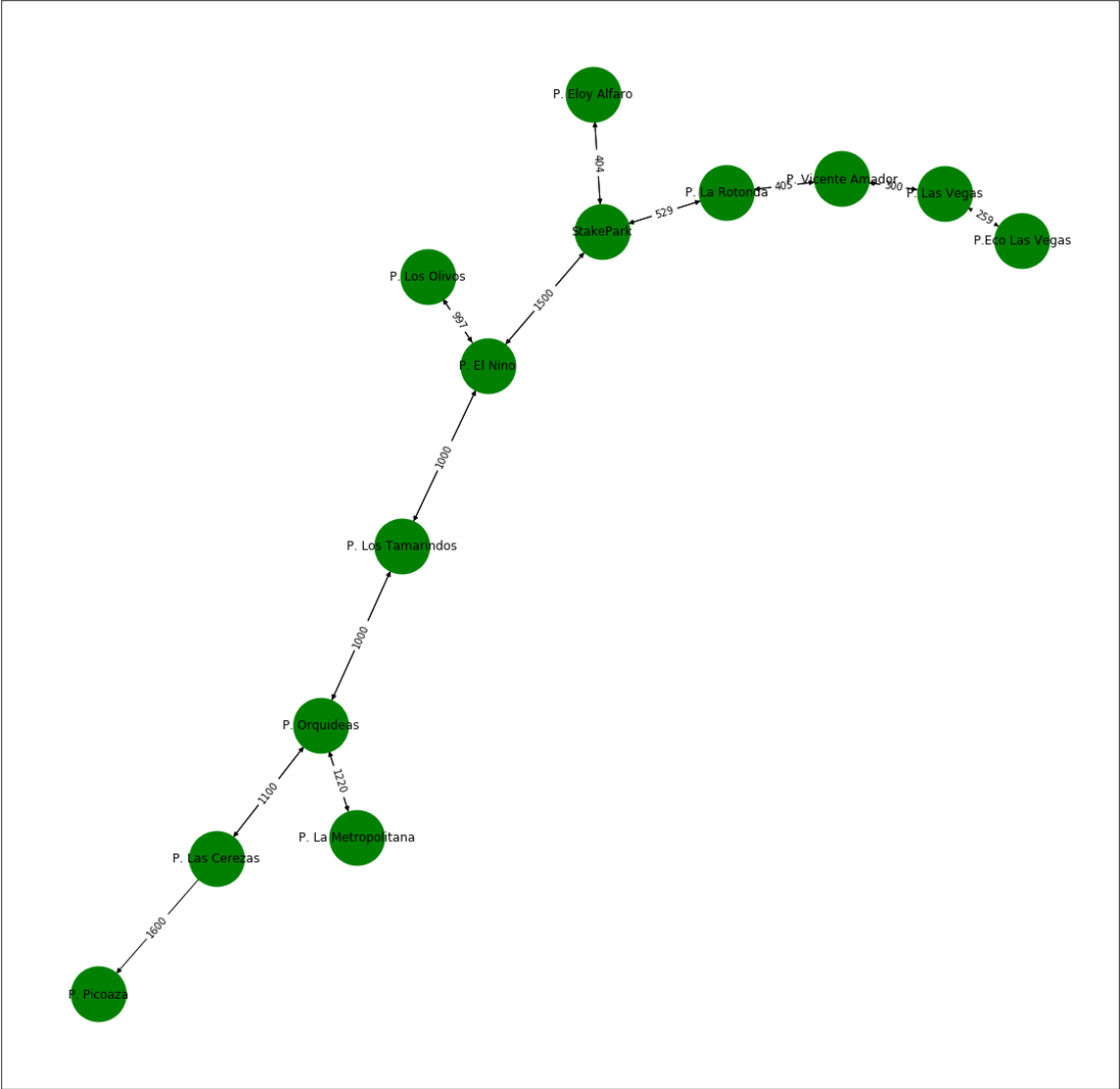
if __name__ == "__main__":
    virus = {
        'P. El Nino': {'P. Los Olivos':997,'P. Los Tamarindos':1000,'StakePark':1500},
        'P. Los Olivos': {'P. El Nino':997},
        'P. Los Tamarindos': {'P. Orquideas':500,'P. El Nino':1000},
        'P. Orquideas': {'P. Las Cerezas':1100,'P. La Metropolitana':1220,'P. Los Tamar
indos':1000},

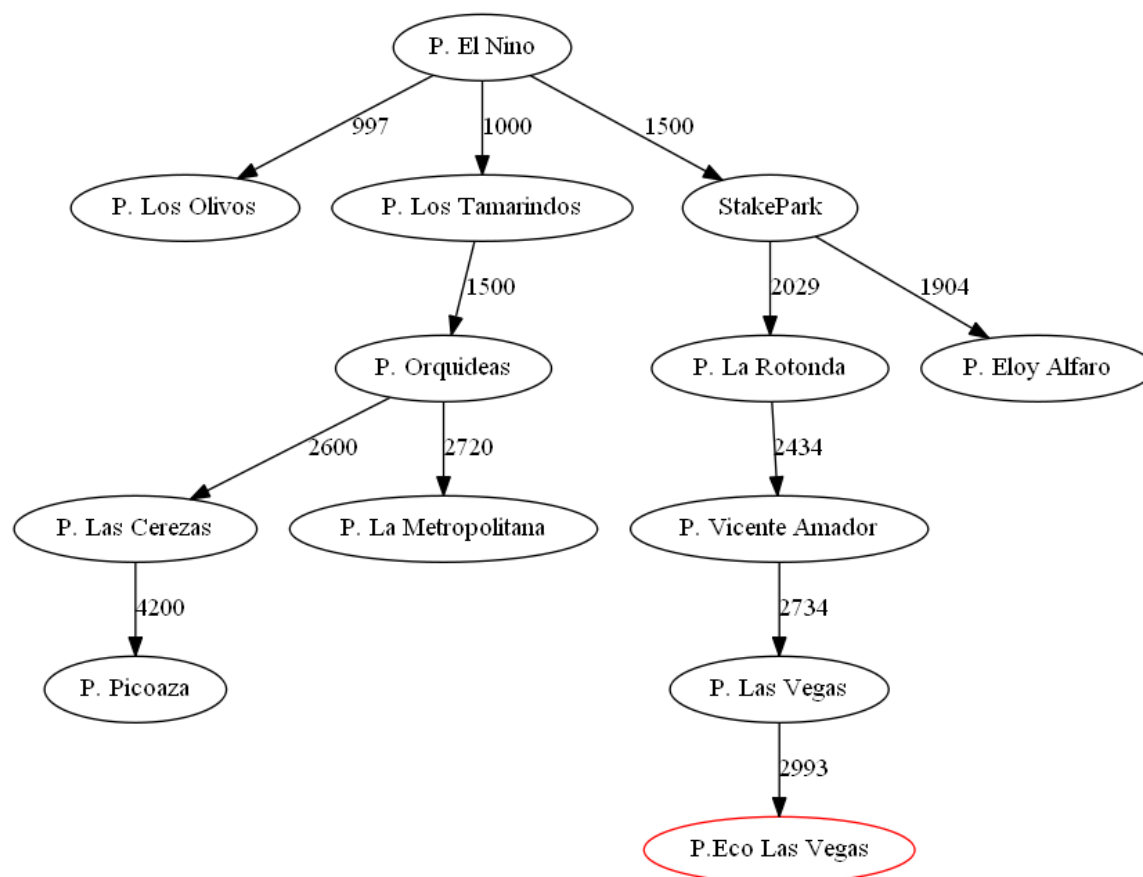
```

```
'P. Las Cerezas': {'P. Picoaza':1600, 'P. Orquideas':1100},
'P. La Metropolitana': {'P. Orquideas':1220},
'StakePark':{'P. La Rotonda':529, 'P. Eloy Alfaro':404, 'P. El Nino':1500},
'P. Eloy Alfaro':{'StakePark':404},
'P. La Rotonda':{'P. Vicente Amador':405, 'StakePark':529},
'P. Vicente Amador':{'P. Las Vegas':300, 'P. La Rotonda':405},
'P. Las Vegas':{'P.Eco Las Vegas':259, 'P. Vicente Amador':300},
'P.Eco Las Vegas':{'P. Las Vegas':259}
}

init_state = 'P. El Nino'
solution = 'P.Eco Las Vegas'
solution_node = search_costo_solucion(virus, init_state, solution, Grafo)
# mostrar resultado
result = []
node = solution_node
if node is not None:
    while node.fathr is not None:
        result.append(node.data)
        node = node.fathr
    result.append(init_state)
    result.reverse() # Reverso el resultado (Solo para presentar)
    print(result)
    print("Costo total: %s" % str(solution_node.cost)) # Imprimir el costo total de
Llegar al nodo
else:
    print("No hay solucion !!!!")
graficar(virus)
graficarNodoResultado(Grafo)
```


['P. El Nino', 'StakePark', 'P. La Rotonda', 'P. Vicente Amador', 'P. Las Vegas', 'P.Eco Las Vegas']
Costo total: 2993





PROFUNDIDAD

In [3]:

```

import networkx as nx
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import Image, display
class Node:
    def __init__(self, data, child=None): # Constructor de la clase
        self.data = data
        self.child = None
        self.fathr = None
        self.cost = None
        self.set_child(child)

    def set_child(self, child): # Agregar hijos
        self.child = child
        if self.child is not None:
            for ch in self.child:
                ch.fathr = self

    def equal(self, node): # Igual al equals de Java
        if self.data == node.data:
            return True
        else:
            return False

    def on_list(self, node_list): # Verfiicar su el nodo esta en la lista
        listed = False
        for n in node_list:
            if self.equal(n):
                listed = True
        return listed

    def __str__(self): # Igual al toString Java
        return str(self.data)

def Compare(node):
    return node.cost

def search_profundidad(init_node, solution, visited, con, costo, g):
    visited.append(init_node.data) #Lista de visitados
    if init_node.data == solution: # Condicion de salida recursividad (Encontro la solu
cion)
        init_node.cost=round(costo,2)
        g.add_node(init_node.data,color='red')
        return init_node # Retorno el nodo resultado
    else:
        # Expandir nodos sucesores (hijos)
        node_data = init_node.data
        child_list=[]
        for node in con[node_data]:
            child = Node(node)
            cost = con[node_data][node]
            child.cost=round(cost,2)
            child_list.append(child)
        child_list = sorted(child_list, key=Compare)
        init_node.set_child(child_list)
        for son in init_node.child: # Recorrer Los nodos hijos
            if not son.data in visited: # No deben estar en Los nodos visitados
                # Llamada Recursiva
                costo=costo+son.cost

```

```

        g.add_edge(init_node.data,son,label=costo)
        Solution = search_profundidad(son, solution, visited,con,costo,g)
        if Solution is not None: # Cuando encuentra una solucion
            return Solution # Retornamos la solucion encontrada
    return None

educativos = {

    'P. El Nino': {'P. Los Olivos':997,'P. Los Tamarindos':1000,'StakePark':1500},
    'P. Los Olivos': {'P. El Nino':997},
    'P. Los Tamarindos': {'P. Orquideas':500,'P. El Nino':1000},
    'P. Orquideas': {'P. Las Cerezas':1100,'P. La Metropolitana':1220,'P. Los Tamarindos':1000},
    'P. Las Cerezas': {'P. Picoaza':1600,'P. Orquideas':1100},
    'P. Picoaza': {'P. Las Cerezas':1600},
    'P. La Metropolitana': {'P. Orquideas':1220},
    'StakePark': {'P. La Rotonda':529,'P. Eloy Alfaro':404,'P. El Nino':1500},
    'P. Eloy Alfaro': {'StakePark':404},
    'P. La Rotonda': {'P. Vicente Amador':405,'StakePark':529},
    'P. Vicente Amador': {'P. Las Vegas':300,'P. La Rotonda':405},
    'P. Las Vegas': {'P.Eco Las Vegas':259, 'P. Vicente Amador':300},
    'P.Eco Las Vegas': {'P. Las Vegas':259}

}

Grafo=nx.DiGraph()

def view_pydot(pdot):
    plt= Image(pdot.create_png())
    display(plt)

def graficarNodo(datos):
    graf = nx.DiGraph()
    graf.add_nodes_from(datos)
    for valor, listaValor in datos.items():
        for a in listaValor:
            graf.add_edge(valor,a,size=250,length=str(listaValor[a]))
    pos = nx.spring_layout(graf)
    plt.figure(figsize=(20,20))
    labels = nx.get_edge_attributes(graf,'length')
    nx.draw_networkx(graf, pos, node_color = 'green', with_labels = True, node_size=300)
    nx.draw_networkx_edge_labels(graf,pos,edge_labels=labels,font_color='black',font_size=10)
    plt.show()

def graficarNodoResultado(grafo):
    p=nx.drawing.nx_pydot.to_pydot(grafo)
    for i, edge in enumerate(p.get_edges()):
        edge.set_label(str(edge.get_label()))
    view_pydot(p)

init_state = 'P. El Nino' # Creamos un estado inicial
solution = 'P.Eco Las Vegas' # La solucion que debe buscar
#Inicializamos las variables
solution_node = None
visited = []
init_node = Node(init_state)
costo = 0

```



```
node = search_profundidad(init_node, solution, visited,educativos, costo, Grafo) # Llamamos al metodo de busqueda
# Mostrar Resultado
result = []
if node is not None:
    fcosto=node.cost
    while node.fathr is not None:
        result.append(node.data)
        node = node.fathr
    result.append(init_state)
    result.reverse() # Reverso el resultado (Solo para presentar)
    print(result)
    print("Costo total: %s" % str(fcosto)) # Imprimir el costo total de llegar al nodo
else:
    print("No hay solucion")
```

```
['P. El Nino', 'StakePark', 'P. La Rotonda', 'P. Vicente Amador', 'P. Las Vegas', 'P.Eco Las Vegas']
Costo total: 5394
```