CSE 505: Adavance Artificial Intelligence ¶

Instructor : [Dr. Chandra Prakash]

Due Date: 20-Jan-2023

In [3]:
```
# Print your name and Roll No.
name=input("Enter your name")
print("My name is : ",name)

# Print the curent date and time
from datetime import datetime
now=datetime.now()
now=now.strftime("%d/%m/%Y %H:%M:%S")
print("Current Date and time is : ",now)
```

```
Enter your nameAnkita Hora
My name is :  Ankita Hora
Current Date and time is :  20/01/2023 12:48:14
```

Assignment 1: Introduction to Python

Excercise 1: Write a python program to input 5 subject marks and calculate total marks, percentages and grades based on the following criteria

i)percentage less than 50 (Grade C)

ii)percentage equal to 50 and less than 80 (Grade B)

iii)percentage equal to 80 and more than 80 (Grade A)

In [11]:
```
#write your code here.
marks=[]
for i in range(5):
    marks.append(int(input("Enter the marks of "+str(i+1)+" subject ")))
print(marks)
```

```
Enter the marks of 1 subject 23
Enter the marks of 2 subject 45
Enter the marks of 3 subject 67
Enter the marks of 4 subject 90
Enter the marks of 5 subject 12
[23, 45, 67, 90, 12]
```

In [59]:
```
total_marks=sum(marks)
print("Total marks ",total_marks)
percent=(total_marks/(100*5))*100
print("Percentage : "+str(percent)+"%")
if(percent<50):
    print("Grade C")
elif(percent>=50 and percent<80):
    print("Grade B")
else:
    print("Grade A")
```

```
Total marks  237
Percentage : 47.4%
Grade C
```

Excercise 2: List operations. Take ten numbers as input from the user and save into a list

i) Write a python program to find the maximum and minimum number in a list of 10 elements and also find the index position of the following numbers.

ii) Write a python program to sort array elements in ascending/descending order using your own defined function.

In [62]:
```
#write your code here.
l1=[]
for i in range(10):
    l1.append(int(input("Enter the "+str(i+1)+" number ")))
print(l1)
```

```
Enter the 1 number 12
Enter the 2 number 43
Enter the 3 number 67
Enter the 4 number 8
Enter the 5 number 90
Enter the 6 number 123
Enter the 7 number 31
Enter the 8 number 64
Enter the 9 number 99
Enter the 10 number 25
[12, 43, 67, 8, 90, 123, 31, 64, 99, 25]
```

In [63]:
```
def max_min(l1,i):
    global max_ind
    global min_ind
    global max1
    global min1


    if(i==len(l1)):
        return

    if(l1[i]>max1):
        max1=l1[i]
        max_ind=i

    if(l1[i]<min1):
        min1=l1[i]
        min_ind=i

    max_min(l1,i+1)

def sort(l1):
    for i in range(len(l1)):
        flag=0
        for j in range(len(l1)-i-1):
            if(l1[j]>l1[j+1]):
                flag=1
                temp=l1[j]
                l1[j]=l1[j+1]
                l1[j+1]=temp
        if(flag==0):
            return
```

In [64]:
```
import math
max1=-math.inf
min1=math.inf
i=0
max_ind=i
min_ind=i
max_min(l1,i)
print("Maximum_no: "+str(max1)+" and index: "+str(max_ind))
max_min(l1,i)
print("Minimum_no: "+str(min1)+" and index: "+str(min_ind))
sort(l1)
print(l1)
```

```
Maximum_no: 123 and index: 5
Minimum_no: 8 and index: 3
[8, 12, 25, 31, 43, 64, 67, 90, 99, 123]
```

Excercise 3: Write a python program to find the factorial of a number

In [42]:
```
def factorial(num,fact):
    if(num==1):
        fact=1
        return fact

    fact=fact*num*factorial(num-1,fact)
    return fact
```

In [47]:

```
#write your code here.
num=int(input("Enter a number "))
fact=1
print("Factorial of a number is ",factorial(num,fact))
```

```
Enter a number 6
Factorial of a number is  720
```

Exercise 4: Create a python function that takes in a graph represented as an adjacency matrix and a starting vertex, and uses depth-first search (DFS) to print all the vertices in the graph in the order they were visited.

Follow the following steps to solve the problem:

Step 1: Understand the problem: Understand the problem statement, the input and output format, and the constraints of the problem.

Step 2: Create the function: Create a python function called "dfs_traversal" that takes in two parameters: an adjacency matrix "graph" representing the graph and an integer "start" representing the starting vertex.

Step 3: Initialize a stack: Initialize an empty stack and push the starting vertex onto the stack.

Step 4: Create a visited list: Create an empty list called "visited" to keep track of the vertices that have been visited.

Step 5: Traverse the graph: Use a while loop to traverse the graph. In each iteration, pop a vertex from the top of the stack, add it to the "visited" list, and push all its unvisited neighbours onto the stack. The neighbours of a vertex can be found by checking the corresponding row of the adjacency matrix.

Step 6: Print the visited list: Print the "visited" list after the while loop has finished executing to show the order in which the vertices were visited.

In [48]:

```
class Edge:
    def __init__(self,src,dest,wt):
        self.src=src
        self.dest=dest
        self.wt=wt
```

In [50]:

```
#write your code here.
vertices=int(input("Enter the number of vertices "))
edges=int(input("Enter the number of edges "))
graph={}
for i in range(vertices):
    graph[i]=[]
for i in range(edges):
    values=list(map(int,input("Enter the src vertex, dest vertex, weight of the edge separated by space ").split(" ")))
    e1=Edge(values[0],values[1],values[2])
    e2=Edge(values[1],values[0],values[2])
    graph[e1.src].append(e1)
    graph[e2.src].append(e2)

print("----------------------------")
print(graph)
```

```
Enter the number of vertices 7
Enter the number of edges 8
Enter the src vertex, dest vertex, weight of the edge separated by space 0 1 10
Enter the src vertex, dest vertex, weight of the edge separated by space 0 2 10
Enter the src vertex, dest vertex, weight of the edge separated by space 1 3 10
Enter the src vertex, dest vertex, weight of the edge separated by space 2 3 20
Enter the src vertex, dest vertex, weight of the edge separated by space 2 4 10
Enter the src vertex, dest vertex, weight of the edge separated by space 4 5 20
Enter the src vertex, dest vertex, weight of the edge separated by space 5 6 10
Enter the src vertex, dest vertex, weight of the edge separated by space 4 6 10
----------------------------
{0: [<__main__.Edge object at 0x000001740C675DF0>, <__main__.Edge object at 0x000001740C76C280>], 1: [<__main__.Edge object
at 0x000001740C66C580>, <__main__.Edge object at 0x000001740C76C520>], 2: [<__main__.Edge object at 0x000001740C76C1C0>, <__
main__.Edge object at 0x000001740C76C040>, <__main__.Edge object at 0x000001740C694EE0>], 3: [<__main__.Edge object at 0x000
001740C76C4F0>, <__main__.Edge object at 0x000001740C76C3D0>], 4: [<__main__.Edge object at 0x000001740C694970>, <__main__.E
dge object at 0x000001740C694760>, <__main__.Edge object at 0x000001740C694460>], 5: [<__main__.Edge object at 0x000001740C6
94FA0>, <__main__.Edge object at 0x000001740C694F40>], 6: [<__main__.Edge object at 0x000001740C6944F0>, <__main__.Edge obje
ct at 0x000001740C694580>]}
```

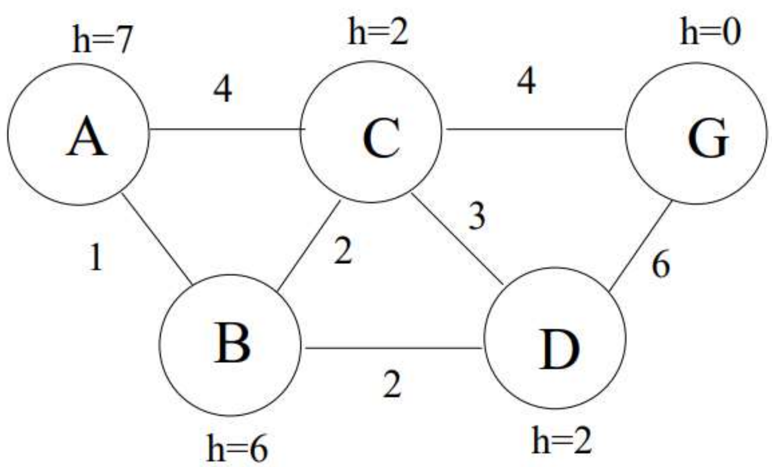
In [56]:

```
def dfs_traversal(graph,i):
    if(visited[i]==True):
        return
    visited[i]=True
    print(visited)
    for neighbors in graph[i]:
        if(visited[neighbors.dest]==False):
            dfs_traversal(graph,neighbors.dest)
    stack.append(i)
```

In [58]:
```
visited=[False]*vertices
print("The visited array after each execution ")
print(visited)
stack=[]
for i in range(vertices):
    if(visited[i]==False):
        dfs_traversal(graph,i)
while(len(stack)!=0):
    rem=stack.pop()
    print(rem)
```

```
The visited array after each execution
[False, False, False, False, False, False, False]
[True, False, False, False, False, False, False]
[True, True, False, False, False, False, False]
[True, True, False, True, False, False, False]
[True, True, True, True, False, False, False]
[True, True, True, True, True, False, False]
[True, True, True, True, True, True, False]
[True, True, True, True, True, True, True]
0
1
3
2
4
5
6
```

Exercise 5: A graph to be searched, starting at A and ending at G. The h values are the heuristic estimates, and the numbers on the edges are the actual costs. Assume that the children of a node are ordered in alphabetical order; also use the alphabetical order to break ties, if necessary.



A). Perform a depth-first search, without using any visited or expanded lists. Show the sequence of expanded nodes. Which path is returned?

B).Of the search algorithms covered, which one requires the smallest number of expansions before returning a path? Which path is returned?

C). Did A∗search with expanded list return the optimal path? Explain why in terms of the admissibility and/or consistency of the heuristics.

A). Perform a depth-first search, without using any visited or expanded lists. Show the sequence of expanded nodes. Which path is returned?

In [2]:
```
graph={}
for i in range(4):
    graph[chr(ord('A')+i)]=[]
graph['G']=[]
print(graph)
```

```
{'A': [], 'B': [], 'C': [], 'D': [], 'G': []}
```

In [3]:
```
class Edge:
    def __init__(self,src,dest,wt):
        self.src=src
        self.dest=dest
        self.wt=wt
```

In [4]:
```
e=int(input("Enter the no. of edges "))
for i in range(e):
    values=input("Enter the src, dest and weight separated by space in alphabetical order ").split(" ")
    e1=Edge(values[0],values[1],values[2])
    graph[e1.src].append(e1)
print(graph)
```

```
Enter the no. of edges 7
Enter the src, dest and weight separated by space in alphabetical order A B 1
Enter the src, dest and weight separated by space in alphabetical order A C 4
Enter the src, dest and weight separated by space in alphabetical order B C 2
Enter the src, dest and weight separated by space in alphabetical order B D 2
Enter the src, dest and weight separated by space in alphabetical order C D 3
Enter the src, dest and weight separated by space in alphabetical order C G 4
Enter the src, dest and weight separated by space in alphabetical order D G 6
{'A': [<__main__.Edge object at 0x000001F5DA40DCD0>, <__main__.Edge object at 0x000001F5DA40DBB0>], 'B': [<__main__.Edge obj
ect at 0x000001F5DA40DDF0>, <__main__.Edge object at 0x000001F5DA40DE20>], 'C': [<__main__.Edge object at 0x000001F5DA40DEE0
>, <__main__.Edge object at 0x000001F5DA40DF40>], 'D': [<__main__.Edge object at 0x000001F5DA40DF70>], 'G': []}
```

In [17]:
```
def DFS(graph,src,dest):

    if src in stack:
        return

    if(src==dest):
        stack.append(src)
        return


    for neighbors in graph[src]:
        if(ord(neighbors.dest)>ord(src)):
            DFS(graph,neighbors.dest,dest)
    stack.append(src)
```

In [18]:
```
src='A'
dest='G'
stack=[]
DFS(graph,src,dest)
while(len(stack)!=0):
    rem=stack.pop()
    print(rem)
```

```
A
B
C
D
G
```

A* algorithm

In [38]:
```
class Edge:
    def __init__(self,src,dest,wt):
        self.src=src
        self.dest=dest
        self.wt=wt
```

In [39]:
```
graph={}
v=int(input("Enter the no. of vertices : "))
for i in range(v-1):
    graph[chr(ord('A')+i)]=[]
graph['G']=[]
print(graph)
```

```
Enter the no. of vertices : 5
{'A': [], 'B': [], 'C': [], 'D': [], 'G': []}
```

In [40]:
```
heu={}
for elem in graph:
    heu[elem]=int(input("Enter the heuristic value for "+elem+": "))
print(heu)
e=int(input("Enter the no. of edges "))
for i in range(e):
    values=input().split(" ")
    e1=Edge(values[0],values[1],values[2])
    e2=Edge(values[1],values[0],values[2])
    graph[e1.src].append(e1)
    graph[e2.src].append(e2)
print(graph)
```

```
Enter the heuristic value for A: 7
Enter the heuristic value for B: 6
Enter the heuristic value for C: 2
Enter the heuristic value for D: 2
Enter the heuristic value for G: 0
{'A': 7, 'B': 6, 'C': 2, 'D': 2, 'G': 0}
Enter the no. of edges 7
A B 1
A C 4
B C 2
B D 2
C D 3
C G 4
D G 6
{'A': [<__main__.Edge object at 0x000001DBE7CD9940>, <__main__.Edge object at 0x000001DBE88C61F0>], 'B': [<__main__.Edge obj
ect at 0x000001DBE7E88850>, <__main__.Edge object at 0x000001DBE88C62E0>, <__main__.Edge object at 0x000001DBE88C62B0>],
'C': [<__main__.Edge object at 0x000001DBE88C6250>, <__main__.Edge object at 0x000001DBE88C6310>, <__main__.Edge object at 0
x000001DBE878DB20>, <__main__.Edge object at 0x000001DBE878DD30>], 'D': [<__main__.Edge object at 0x000001DBE88C6370>, <__ma
in__.Edge object at 0x000001DBE878DAF0>, <__main__.Edge object at 0x000001DBE878DC10>], 'G': [<__main__.Edge object at 0x000
001DBE878D0D0>, <__main__.Edge object at 0x000001DBE878DC40>]}
```

In [41]:
```
class Pair:
    def __init__(self,vertex,path,cost):
        self.vertex=vertex
        self.path=path
        self.cost=cost

    def __lt__(self,other):
        return self.path < other.path
```

In [42]:
```
def A_star_algo(graph,src,path,pq,visited,cost,dest):
    pq.put((heu[src],Pair(src,path+src,cost)))

    while(not pq.empty()):
        rem=pq.get()
        if(rem[1].vertex==dest):
            visited[rem[1].vertex]=True
            print(rem[1].path)
            return

        if(visited[rem[1].vertex]==False):
            visited[rem[1].vertex]=True
            for neighbors in graph[rem[1].vertex]:
                if(visited[neighbors.dest]==False):
                    pq.put((rem[1].cost+int(neighbors.wt)+heu[neighbors.dest],
                            Pair(neighbors.dest,rem[1].path+neighbors.dest,rem[1].cost+int(neighbors.wt))))
```

In [43]:
```
visited={}
for elem in graph:
    visited[elem]=False
print(visited)
from queue import PriorityQueue
pq=PriorityQueue()
print(pq)

dest=input("Enter the destination : ")

for elem in graph:
    if(visited[elem]==False):
        path=""
        cost=0
        A_star_algo(graph,elem,path,pq,visited,cost,dest)
```

```
{'A': False, 'B': False, 'C': False, 'D': False, 'G': False}
<queue.PriorityQueue object at 0x000001DBE879C0A0>
Enter the destination : G
ACG
```

BFS

In [47]:
```
class Pair:
    def __init__(self,vertex,path):
        self.vertex=vertex
        self.path=path
```

In [48]:
```
def BFS(graph,visited,src,dest):
    queue.append(Pair(src,""+src))
    while(len(queue)!=0):
        rem=queue.pop(0)
        if(rem.vertex==dest):
            visited[rem.vertex]=True
            print(rem.path)
            return

        if(visited[rem.vertex]==False):
            visited[rem.vertex]=True
            for neighbors in graph[rem.vertex]:
                if(visited[neighbors.dest]==False):
                    queue.append(Pair(neighbors.dest,rem.path+neighbors.dest))
```

In [49]:
```
queue=[]
for elem in graph:
    visited[elem]=False
dest=input("Enter the destination : ")
for elem in graph:
    if(visited[elem]==False):
        BFS(graph,visited,elem,dest)
```

```
Enter the destination : G
ACG
```

Uniform cost search

In [56]:
```
class Pair:
    def __init__(self,vertex,path):
        self.vertex=vertex
        self.path=path

    def __lt__(self,other):
        return self.path < other.path
```

In [57]:
```
def UCS(graph,visited,src,dest,pq):
    pq.put((0,Pair(src,""+src)))
    while(not pq.empty()):
        rem=pq.get()
        if(rem[1].vertex==dest):
            visited[rem[1].vertex]=True
            print(rem[1].path)
            return

        if(visited[rem[1].vertex]==False):
            visited[rem[1].vertex]=True
            for neighbors in graph[rem[1].vertex]:
                if(visited[neighbors.dest]==False):
                    pq.put((rem[0]+int(neighbors.wt),Pair(neighbors.dest,rem[1].path+neighbors.dest)))
```

In [58]:
```
from queue import PriorityQueue
pq=PriorityQueue()
for elem in graph:
    visited[elem]=False
dest=input("Enter the destination : ")
for elem in graph:
    if(visited[elem]==False):
        UCS(graph,visited,elem,dest,pq)
```

```
Enter the destination : G
ABCG
```

Best First Search algo

In [63]:
```
class Pair:
    def __init__(self,vertex,path):
        self.vertex=vertex
        self.path=path

    def __lt__(self,other):
        return self.path < other.path
```

In [64]:
```
def Best_first_Search(graph,visited,src,dest,pq):
    pq.put((heu[src],Pair(src,""+src)))
    while(not pq.empty()):
        rem=pq.get()
        if(rem[1].vertex==dest):
            visited[rem[1].vertex]=True
            print(rem[1].path)
            return

        if(visited[rem[1].vertex]==False):
            visited[rem[1].vertex]=True
            for neighbors in graph[rem[1].vertex]:
                if(visited[neighbors.dest]==False):
                    pq.put((heu[neighbors.dest],Pair(neighbors.dest,rem[1].path+neighbors.dest)))
```

In [65]:
```
from queue import PriorityQueue
pq=PriorityQueue()
for elem in graph:
    visited[elem]=False
print("Heuristic values are : ",heu)
dest=input("Enter the destination : ")
for elem in graph:
    if(visited[elem]==False):
        Best_first_Search(graph,visited,elem,dest,pq)
```

```
Heuristic values are :  {'A': 7, 'B': 6, 'C': 2, 'D': 2, 'G': 0}
Enter the destination : G
ACG
```

B).Of the search algorithms covered, which one requires the smallest number of expansions before returning a path? Which path is returned?

A* algorithm, BFS Algo and Best First Search Algo required the smallest number of expansions before returning a path. The path returned was "ACG"

C). Did A∗search with expanded list return the optimal path? Explain why in terms of the admissibility and/or consistency of the heuristics.

No, A* algorithm did not returned the optimal path because the total cost to reach the goal with this path is 8 which is more than the optimal path cost to reach the goal . It is not optimal because it is not consistent . It is only admissible. It is not consistent because it does not satisfy the property that for every node n, every successor n' of n generated by any action a, $h(n) \leq c(n,a,n') + h(n')$