CCT College Dublin Continuous Assessment

Module Title:	Advanced Numerical Methods
Assessment Title:	CA3
Lecturer Name:	John O’Sullivan
Student Full Name:	Leisly Alitzel Pino Duran
Student Number:	2020303
Assessment Due Date:	11th of December 2022
Date of Submission:	11th of December 2022

Declaration

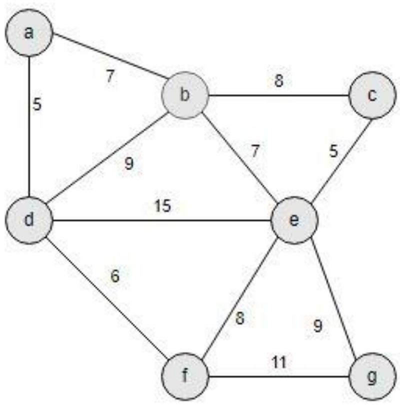
Table of contents

Prim's algorithm

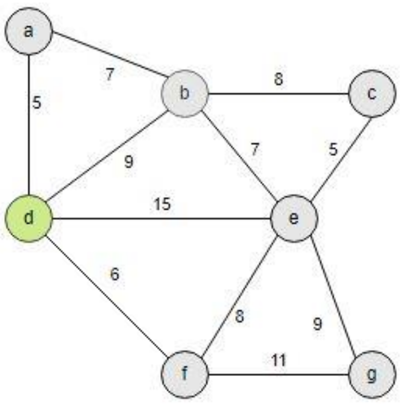
A minimum spanning tree in a connected weighted graph is a spanning tree with the smallest possible sum of weights of its edges. This helps to solve problems that can arise in different industries; an example of this can be in telephone networks, to find the shortest way to communicate several telephone towers. Another elementary example is a traveller wanting to find the most economical way to visit all the desired cities.

Prim's algorithm was discovered in 1930 by mathematician Vojtech Jarnik and later independently by computer scientist Robert C. Prim in 1957. Because of this, it is known as Prim's algorithm (and sometimes as the Prim-Jarník algorithm). Begin by choosing any edge with the most negligible weight and putting it into the spanning tree. Successively add to the tree edges of minimum weight that are incident to a vertex already in the tree, never forming a simple circuit with those edges already in the tree.

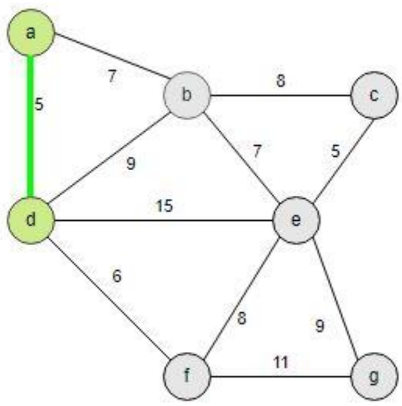
An example of this type of algorithm is the following. First, we declare an array named: closed list and consider the available list as a priority queue with min-heap. Adding a node and its edge to the closed list indicates that we have found an edge that links the node into the minimal spanning tree. As a node is added to the closed list, its successors (immediately adjacent nodes) are examined and added to the priority queue of open nodes.



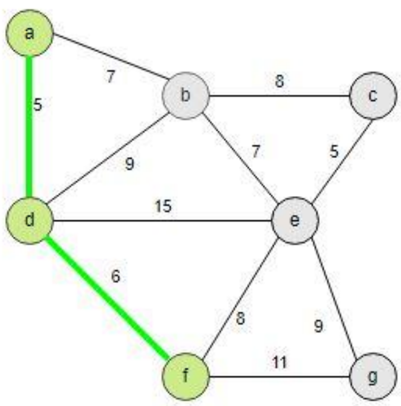
Open list: **d**
Close list:



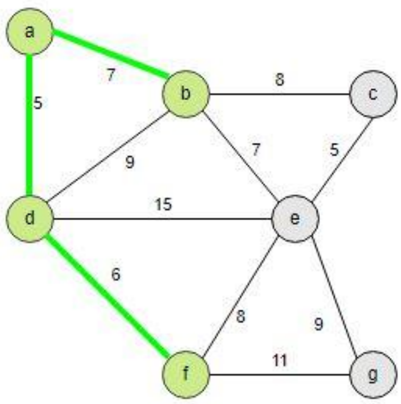
Open list: **a, f, e, b**
Close list: **d**



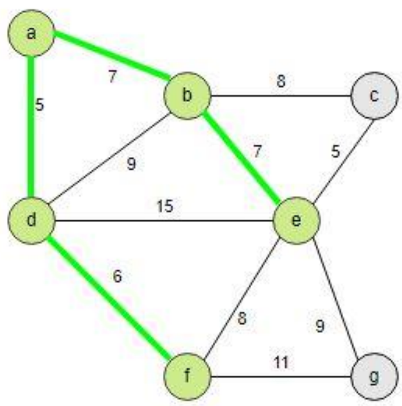
Open list: **f, e, b**
Close list: **d, a**



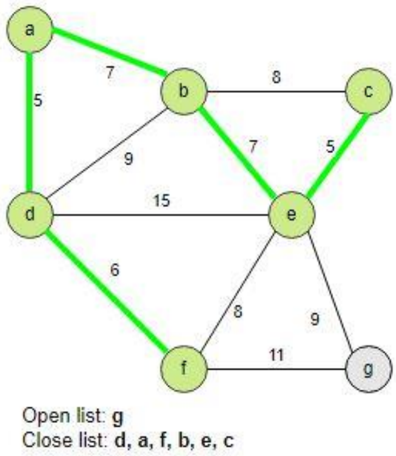
Open list: **b, e, g**
Close list: **d, a, f**

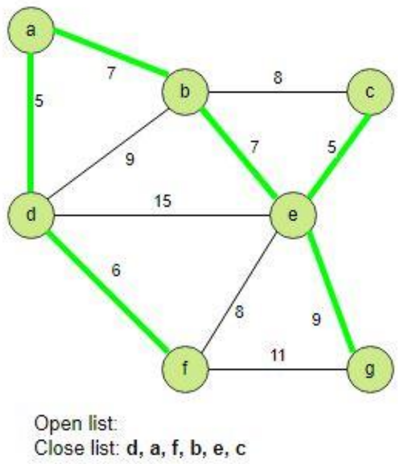


Open list: **e, g, c**
Close list: **d, a, f, b**



Open list: **c, g**
Close list: **d, a, f, b, e**





The length of the minimum spanning tree is 5+6+7+7+5+9=39

Algorithm:

1. Choose any starting vertex. Look at all edges connecting to the chosen starting vertex, choose the edge with the lowest weight, and add this to the tree.

2. Look at all the edges connected to the tree. Choose the edge with the lowest weight connected to the chosen vertices and add that to the tree. If an edge is already connected to a chosen vertex, we do not consider it.

3. Repeat step 2 until all vertices are in the tree.

PseudoCode:

1. Make a queue (Q) with all the vertices of G (V);

2. For each member of Q, set the priority to INFINITY;

3. Only for the starting vertex(s) set the priority to 0;

4. The parent of (s) should be NULL;

5. While Q is not empty

6. Get the minimum from Q – let us say (u); (priority queue);

7. For each adjacent vertex to (v) to (u)

8. If (v) is in Q and weight of (u, v) < priority of (v), then

9. The parent of (v) is set to be (u)

10. The priority of (v) is the weight of (u, v)

In conclusion, using this type of algorithm is due to time complexity. Initially, this type of problem starts having $O(V^2)$, giving an advantage in time to comply faster if a list represents the input graph of adjacency, the time complexity of Prim's algorithm can be reduced to O(E log V).

Probability:

1. What is the probability of rolling exactly two 6s in five rolls of a fair die?

To find the result, the binomial equation was applied where it can be determined:

n= 5 k=2 p=$\frac{1}{6}$

and add the data of the following formula:

$$P(x = k) = \binom{n}{k} p^k (1-p)^{n-k} \qquad P(x = k) = \binom{5}{2} \left(\frac{1}{6}\right)^2 \left(\frac{5}{6}\right)^3 = 0.16$$

Calculations made in python, see Appendix 1

2. The number of industrial injuries on average per working week in a factory is 0.75. Assuming that the distribution of injuries follows a Poisson distribution, find the probability that in a particular week there will be no more than two accidents.

To find the result, the poisson equation was applied where it can be determined:

k=2 λ=0.75

and add the data of the following formula:

$$P(x \leq k) = \frac{\lambda^k e^{-\lambda}}{k!} \qquad P(x \leq k) = \frac{0.75^2 e^{-0.75}}{2!} \quad = 0.96$$

Calculations made in python, see Appendix 1

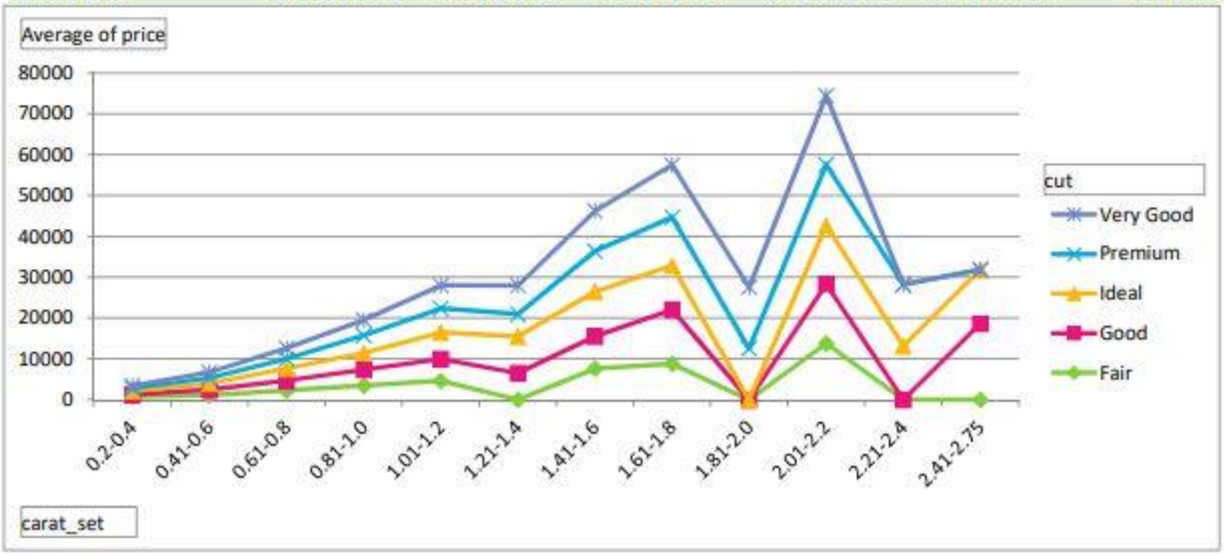
Statistics:

The analysis of the Diamonds dataset was done using Python and Excel. In Graph 1, the price of diamonds was analyzed based on the carat and the type of cut. Before plotting the data, we expected to find that the higher the carat and the better type of cut, the higher the price of the diamond would be. We select several carat ranges to group the data and obtain the average price for the data analysis through the graph. We concluded from the analysis that higher carats and better cuts were less positively correlated with a higher price than expected. We found the highest-priced diamonds were between 2.2 and 2.4 carats with a very good cut. Furthermore, this is verified with the calculations made in Python of the Variance and the standard deviation.

Regarding Graph 2, we discovered that the colour with higher market Prices is D and J.(clarities I1 and VVS2). However, the higher price for the other 5 (E, F, G, H, I) colours was mainly found in the diamond's clarity SI2. Regarding the diamonds with the lower price, we found that the clarity VVS1 was usually the cheapest regardless of the colour. This explanation can be verified in Appendix 2.

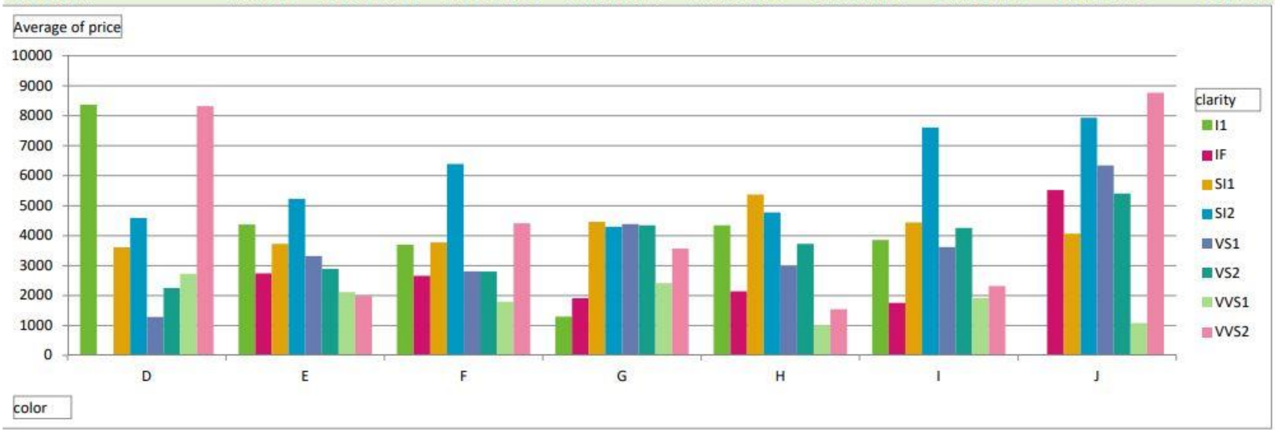
Graph 1:

Average of price	Column Labels					
Row Labels	Fair	Good	Ideal	Premium	Very Good	Grand Total
0.2-0.4	743	559.8571429	790.6976744	742.9714286	639.8305085	733.8424908
0.41-0.6	1118.875	1293	1565.510417	1388.4	1408.0625	1469.31694
0.61-0.8	2273.2	2447.111111	2947.857143	2352.407407	2613.577778	2686.114094
0.81-1.0	3498	3887	4065.866667	4317.421053	3806.791667	3958.901235
1.01-1.2	4634	5364.363636	6524.473684	5858.311111	5622.605263	6005.394737
1.21-1.4		6556.666667	8914.647059	5492.916667	7027.25	7305.931818
1.41-1.6	7641.5	7882.333333	10932.11765	9994.238095	9802.6875	9921.693548
1.61-1.8	8800	13287	10790.83333	11854.6	12773	11723.31579
1.81-2.0				12585.5	15002.33333	14035.6
2.01-2.2	13831	14534.5	14315.8	14940	16852.6	15076.52
2.21-2.4			13200	15059.5		14687.6
2.41-2.75		18686	13156			15921
Grand Total	**3765.107143**	**4119.304348**	**3368.457002**	**4302.851563**	**3968.829167**	**3814.666**



Graph 2:

Average of price	Column Labels								
Row Labels	I1	IF	SI1	SI2	VS1	VS2	VVS1	VVS2	Grand Total
D	8372.5		3607.40625	4589.275862	1284	2246.184211	2728	8323.3	3529.977099
E	4371	2738.714286	3727.826087	5229.307692	3316.5	2891.162791	2110.266667	1992.380952	3330.55
F	3696.666667	2658.333333	3775.482759	6393.555556	2809.16	2800.54902	1791.181818	4408.375	3628.210526
G	1298	1906.416667	4460.133333	4293.666667	4384.166667	4345.8	2408.882353	3568.9	3942.680628
H	4346.8	2142.833333	5368.765957	4773.545455	2973.555556	3725.870968	1009	1544.444444	3847.267045
I	3855	1749.833333	4439.92	7605.5	3618.882353	4257.7	1911.5	2315.875	4319.171717
J		5519	4067.529412	7933.5	6343.583333	5406.466667	1078	8770	5279.75
Grand Total	**4448.375**	**2313.365854**	**4259.681416**	**5381.597484**	**3489.043478**	**3414.884774**	**1944.116667**	**3426.404255**	**3814.666**



References

Applications of Minimum Spanning Tree Problem - GeeksforGeeks [online]. *GeeksforGeeks*. Available in: https://www.geeksforgeeks.org/applications-of-minimum-spanning-tree/

Prim’s Minimum Spanning Tree (MST) | Greedy Algo-5 - GeeksforGeeks [online], (sin fecha). *GeeksforGeeks*. Available in: https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/

Rosen, K. H., (2003). *Discrete mathematics and its applications*. 5ª ed. Boston: McGraw-Hill.

Schwitzgebel, E., (2009). *Introduction to Algorithms*. Cambridge, Mass.: The MIT Press.

Advanced Numerical Methods

Continuous Assessment 3

Probability

Lecturer: John O'Sullivan

Student: Leisly Alitzel Pino Duran

Student Number: 2020303

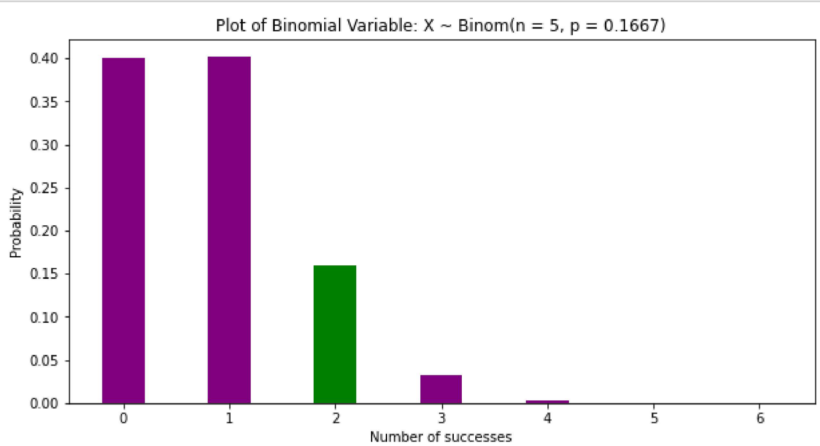
In [1]:
```
#Importing libaries

import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import binom
from scipy.stats import poisson
```

1. What is the probability of rolling exactly two 6s in five rolls of a fair die?

In [2]:
```
# Calculate binomial probability P(X = 2) when X ~ Binom(n = 5, p = 0.1667)

result= round(binom.pmf(k=2, n=5, p=0.167), 2)
result
```

Out[2]: 0.16

In [3]:
```
# Creating a plot:

fig = plt.figure(figsize = (10, 5))

# Printing all probabilities:

x = np.array([0, 1, 3, 4, 5, 6])
outpro = binom.pmf(k=x, n=5, p=0.167)

# Creating the bar plot

plt.bar(x, outpro, color ='purple', width = 0.4)
plt.bar(2, result, color ='green', width = 0.4)

plt.xlabel("Number of successes")
plt.ylabel("Probability")
plt.title("Plot of Binomial Variable: X ~ Binom(n = 5, p = 0.1667)")
plt.show()
```



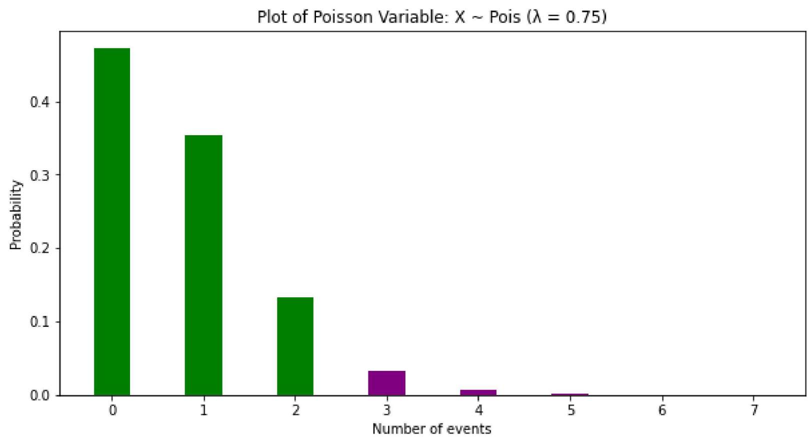
2. The number of industrial injuries on average per working week in a factory is 0.75. Assuming that the distribution of injuries follows a Poisson distribution, find the probability that in a particular week there will be no more than two accidents.

In [4]:
```
# Calculate poisson probability P (X <=2 ) when X ~ Pois(lamdba = 0.75)

result2= round(poisson.cdf(k=2, mu=0.75), 2)
result2
```

Out[4]: 0.96

In [5]:

```
# Creating a plot:

fig = plt.figure(figsize = (10, 5))

# Printing all probabilities:

x2 = np.array([3, 4, 5, 6, 7])
outpro2 = poisson.pmf(k=x2, mu=0.75)
x3 = np.array([0 , 1, 2])
outpro3 = poisson.pmf(k=x3, mu=0.75)

# Creating the bar plot
plt.bar(x2, outpro2, color ='purple', width = 0.4)
plt.bar(x3, outpro3, color ='green', width = 0.4)

plt.xlabel("Number of events")
plt.ylabel("Probability")
plt.title("Plot of Poisson Variable: X ~ Pois (λ = 0.75)")
plt.show()
```



Advanced Numerical Methods

Continuous Assessment 3

Statistics

Lecturer: John O'Sullivan

Student: Leisly Alitzel Pino Duran

Student Number: 2020303

The *diamonds* dataset contains the following variables:

price: Price between 351 - 18 797

carat: Weight of the diamond 0.20 - 2.75

cut: Quality of the cut Fair, Good, Very Good, Premium, Ideal

color: Diamond colour, from J (worst) to D (best)

clarity: A measurement of how clear the diamond is I1 (worst), SI2, SI1, VS2, VS1, VVS2, VVS1, IF (best)

x: Length in mm

y: Width in mm

z: Depth in mm

depth: :The height of a diamond, measured from the culet to the table, divided by its average girdle diameter

table: The width of the diamond's table expressed as a percentage of its average diameter

In [1]:
```
# Importing Libraries:

import pandas as pd
import statistics as stats
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import math
```

In [2]:
```
# Importing the dataset:

dataset = pd.read_csv('diamonds.csv')
```

In [3]:
```
# Look at the top of the dataset:

dataset.head()
```

Out[3]:

	carat	cut	color	clarity	depth	table	price	x	y	z
0	1.14	Premium	F	SI1	61.1	58.0	6448	6.77	6.72	4.12
1	0.80	Very Good	F	VS2	61.3	55.0	3954	5.93	6.01	3.66
2	0.60	Ideal	F	VVS2	60.8	57.0	2856	5.44	5.49	3.32
3	0.71	Premium	H	SI1	62.4	62.0	2207	5.67	5.64	3.53
4	0.90	Good	I	VS2	63.7	61.0	3246	6.10	6.06	3.87

In [4]:
```
# Summary statistics:

dataset.describe()
```

Out[4]:

	carat	depth	table	price	x	y	z
count	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.000000	1000.00000
mean	0.786250	61.760400	57.424200	3814.666000	5.703880	5.706280	3.52386
std	0.467455	1.436961	2.169978	3887.919395	1.117231	1.110027	0.68713
min	0.230000	44.000000	53.000000	351.000000	3.880000	3.840000	2.42000
25%	0.400000	61.100000	56.000000	934.250000	4.720000	4.720000	2.90000
50%	0.700000	61.900000	57.000000	2361.000000	5.680000	5.690000	3.52000
75%	1.040000	62.500000	59.000000	5251.500000	6.530000	6.520000	4.03000
max	2.750000	68.400000	67.000000	18797.000000	9.040000	8.980000	5.54000

In [5]:
```
# Saving the price list in 'price'
price = dataset.price

# Finding the mean
mean = price.mean()
print(round(mean, 2))
```

```
3814.67
```

In [6]:
```
# Finding the median

median = price.median()
print(median)
```

```
2361.0
```

In [7]:
```
# Finding the mode

mode = price.mode()
print(mode)
```

```
0    828
dtype: int64
```

In [8]:
```
# Finding the deviations of the first 10 elements

print("Deviation = ", price[0:10] - mean)
```

```
Deviation =  0    2633.334
1     139.334
2    -958.666
3   -1607.666
4    -568.666
5   -2845.666
6    8171.334
7   -2768.666
8   -2207.666
9   -2897.666
Name: price, dtype: float64
```

In [9]:
```
# Finding the squared deviations of the first 10 elements

print("Squared deviations = ", pow(price[0:10] - mean, 2))
```

```
Squared deviations =  0    6.934448e+06
1    1.941396e+04
2    9.190405e+05
3    2.584590e+06
4    3.233810e+05
5    8.097815e+06
6    6.677070e+07
7    7.665511e+06
8    4.873789e+06
9    8.396468e+06
Name: price, dtype: float64
```

In [10]:
```
# Findind the sum of squared deviations of the entire price list

print("Sum of the squared deviations =", pow(price - mean, 2).sum())
```

```
Sum of the squared deviations = 15100801308.443998
```

In [11]:
```
# Finding the standard deviation

s2 = price.std(ddof=1)
print("Standard deviation =", s2)
```

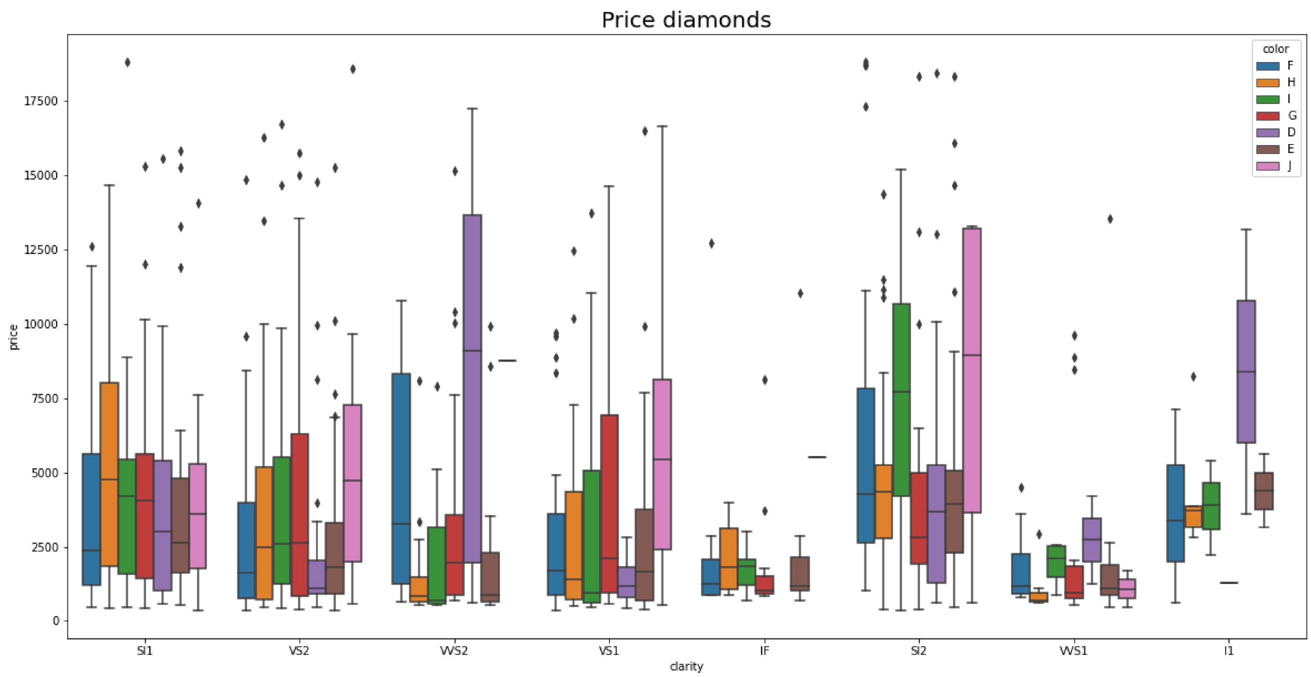
```
Standard deviation = 3887.9193954697257
```

In [12]:
```
# Finding the variance

s_sq2 = price.var(ddof=1)
print("Variance =", s_sq2)
```

```
Variance = 15115917.225669676
```

In [13]:
```
# Creating a boxplot using the variables price, clarity and color

plt.figure(figsize = (20, 10))
sns.boxplot(x= 'clarity', y='price', data=dataset, hue='color')

plt.title('Price diamonds', fontsize=20)
```

Out[13]:
```
Text(0.5, 1.0, 'Price diamonds')
```



In [14]:
```
# Creating a scatterplot using the variables price, carat and cut

sns.scatterplot(x = 'carat', y ='price', data=dataset, hue = "cut")
```

Out[14]:
```
<AxesSubplot:xlabel='carat', ylabel='price'>
```