In [ ]:

```
dice problem
```

In [3]:

```
def printcombo(c,n):
    for i in range(n):
        print(c[i],end="")
    print("")
def generate(d,n,curr,tar):
    if curr==n:
        sum=0
        for i in range(n):
            sum+=d[i]
        if sum==tar:
            printcombo(d,n)
        return
    for i in range(1,6+1):
        d[curr]=i
        generate(d,n,curr+1,tar)
n=2
tar=10
dice={}
generate(dice,n,0,tar)
```

```
46
55
64
```

In [ ]:

```
tsp
```

In [2]:

```
def tsp(graph):
    n=len(graph)
    visiteds=(1<<n)-1
    memo=[[None]*(1<<n) for _ in range(n)]
    def visit(city,visited):
        if visited==visiteds:
            return graph[city][0]
        if memo[city][visited] is not None:
            return memo[city][visited]
        minc = float('inf')
        for nextc in range(n):
            if not visited&(1 << nextc):
                cost=graph[city][nextc]+visit(nextc,visited | (1<<nextc))
                if cost<minc:
                    minc=cost
        memo[city][visited]=minc
        return minc
    return visit(0,1)
graph=[[0,3,2,3],[3,0,2,4],[2,2,0,2],[3,4,2,0]]
shortest=tsp(graph)
print(shortest)
```

```
10
```

In [ ]:

```
obst
```

In [1]:

```
def optcost(freq, i, j):
    if j < i:
        return 0
    if j == i:
        return freq[i]
    fsum = Sum(freq, i, j)
    Min = 10000000
    for r in range(i, j + 1):
        cost = (optcost(freq, i, r - 1) +
                optcost(freq, r + 1, j))
        if cost < Min:
            Min = cost
    return Min + fsum
def optimalSearchTree(keys, freq, n):
    return optcost(freq, 0, n - 1)
def Sum(freq, i, j):
    s = 0
    for k in range(i, j + 1):
        s += freq[k]
    return s
keys = [10,20,30,40]
freq = [2,3,2,4]
n = len(keys)
print("Cost of Optimal BST is", optimalSearchTree(keys, freq, n))
```

```
Cost of Optimal BST is 21
```

In [ ]: