# General Algorithm Backtracking

Algorithm Backtrack (X[1···i])
//Given a template of a generic backtrack algorithm
// Input: X[1···i] Specifies first 'i' promising component of a Solution
//Output: All the Tuples representing the problem's Solution

if (X[1···i]) is a Solution
 Work (X[1···i])

Else
 for each Element $x \in S_{i+1}$ Consistent with X[i···1] and the constraints do

 X[i+1] ⟵ x
 Backtrack(X[i····i+1])

}

20091

# Hamiltonian

## Algorithm Hamiltonian(k)

$TC = O(n^n)$

```
{ repeat
 { nextvalue(k)
  if (X[k]==0) then
    return
  if (k==n)
   write (X[1:n])
  else
  Hamiltonian(k+1)
 } Unn (false)
}
```

## Algorithm Next value(k)

```
{ repeat
 { X[k]=(X[k]+1)%(n+1)
   if (X[k]==0)
    return
  if((G[x[k-1], x[k]] ≠ 0)
  {  for(j=1 to k-1)
     if (X[k]==X[j]
      break;
    if (j==k) then
   if((k<n) or (k==n) and G[X[n], X[1]] ≠ 0)
      return ;
  }}  Until (false)
}
```

# Graph Coloring.

~~//This at~~

Algorithm mColoring(k)

//This algorithm was formed using Recursive backtracking

//The graph is represented as boolean adjacency G[1:n, 1:n]

//k is the index of the next vertex to color

```
{ repeat
  { Nextvalue(k);
    if (X[k] = 0)
      return
    if (k == n) then
      write(x[1:n]);
  Else
    mColoring(k+1);
  } until(false);
}
```

$$TC = O(nm^n)$$

```
Algorithm Nextvalue(k)
{ repeat {
    X[k] = (X[k]+1) mod(n+1)
    if (X[k] = 0)
      return
  for (j = 1 to n) do }
  { if ((G[k,j] ≠ 0) and (x[k] = x[j]))
    then break
  }
    if (j = n+1) then
      return
  } Until(false);
}
```

## Sum of Subsets

$\boxed{\sum w_i x_i , k , \sum w_i}$

$x = 1$: $\boxed{\sum w_i x_i + w_k , k+1 , \sum w_i - w_k}$

$x = 0$: $\boxed{\sum w_i x_i , k+1 , \sum w_i - w_k}$

```
Algorithm Sumq Subset (s, k, r)
{ X[k] = 1
  if (s + w[k] = m)
  then write (X[1:k]);
  else if (s + w[k] + w[k+1] <= m) then
     Sumq subset (s + w[k], k+1, r - w[k]);

  if ((s + r - w[k] >= m) and (s + w[k+1] <= m)) then
  { X[k] = 0
    Sumq Subset (s, k+1, r - w[k]);
  }
}
```

Time Complexity

$TC = O(2^n)$

# Knapsack - backtracking.

Algorithm Bknap(k, cp, cw)

```
{ //left child
  if(cw + w[k] <= m) then
 {  y[k] = 1
  if (k<n) then
 Bknap (k+1, cp+p[k]  , cw +w[k]);
 if((cp + p[k] > fp) and(k=n)) then
{ fp = cp + p[k]
  fw = cw + w[k]
  for( j=1 to k) do
   x[j] = y[j];
}
}
    //Right child
   if (Bound (cp, cw, k) >= fp) then
  {   y[k] = 0
      if (k<n) then
    Bknap (k+1, cp, cw)
    if((cp > fp) and (k=n)) then
    { fp=cp;
     fw = cw;
     for j=1 to k do
      x[j] = y[j]
   }
  }
}
```

Algorithm Bound (cp, cw, k)

{ b = cp ;

c = cw ;

for ( i = k+1 to n do )

{ c = c + w[i]

if ( c < m ) then

b = b + p[i]

Else

return b + (1 - (c - m) / w[i] * p[i] ;

}

return k ;

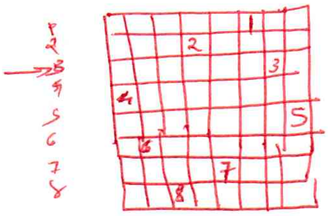
}

Time Complexity TC = $O(2^n)$

# N Queens

4 queens : (2 4 1 3)
(3 1 4 2)

8 queens (4, 6, 8, 2, 7, 1, 3, 5)



```
Algorithm NQueens(k, n)
{ for i := 1 to n do
  { if place(k, i) then
    { x[k] = i
      if (k == n)
        write (X[1:n]);
      Else  Nqueen(k+1, n);
    }
  }
}
```

```
Algorithm Place (k, i)
{ for(j = 1 to k-1)
  { if ((X[j] = i) or (Abs(X[j] - i) = Abs(j-k)))
      then return false
    return True
  }
```

same column

same diagonal

TC = Best Case O(n!)