Backtracking

A backtracking is a systematic method that tries to construct a solution to a computational problem *incrementally*, one small piece at a time.

Whenever the algorithm needs to decide between *multiple alternatives* to the next component of the solution, it recursively iterate through *all the possible configurations* of a search space.

It is a general method that must be customized for the given problem.

In the general case, we will model our solution in a vector X = (X1, X 2, X 3, ... , Xn) where each element Xi is selected from a finite search space for that position. X may present a permutation of some items, a sequence of moves in a game, a path in a maze, or subset of a set.

Backtracking mechanizm start the first position of X (that is X1 or X0), assign the first candidate element for that positon and go on with the next position (that is X2 or X1). At each step it start from the previous partial solution, and try to extend it by adding another element in the current position. In each step we must check if we got the solution, if we got the solution we print it. If not, we must check if the partial solution is still potentially extendible to some complete solution. If so, we recur and continue. If not, se delete the last element from X and try next canditate element for that position.

Backtracking is the main algorithm to solve he exponiential (NP-Hard) problems. It enemurates all possible solutions of a problem.

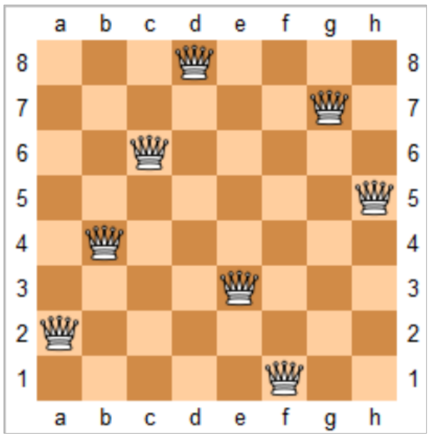
Here is a general template for a backtracking algorithm

```
int N;
vector<int> X;
.
.
.
void back(int i)
{
	if (isASoltuion(i))
	{
		printOneSol();
		return;
	}

	for (int j = 0; j < N; j++)
	{
		X[i] = j;
		if (isValid(i))
			back(i + 1);
	}
}
```

Sample Problem 1: N Quens

The n-queens puzzle is the problem of placing n queens on an n×n chessboard such that no two queens attack each other. Make a program that prints all solutions of this puzzle.



Code:

```
// N Quens.cpp : Backtracking solution.
#include "pch.h"
#include <iostream>
#include <vector>

using namespace std;

int N;                          //size of the board
vector<int> X;                  //X[i] is the queen in the ith row and X[i]th column.
vector<bool> used;              //columns of the previously placed queens.
//-------------------------------------
//Check if the kth element attacks the previous ones
bool isValid(int k)
{
       if (used[X[k]])              //same column
              return false;

       //check for diagonals
       for(int i=0; i<k; i++)
                     if (abs(i-k) == abs(X[i]-X[k]))
                            return false;
       return true;
}
//-------------------------------------
void printOneSolution()
{
       for (int i = 0; i < N; i++)
              cout << "(" << i << ", " << X[i] << ") ";
       cout << endl;
}
//-------------------------------------
void back(int i)
{
       //If the fist N columns are occupied, we have a new solution
       if (i == N)
       {
              printOneSolution();
              return;
       }

       //Candidate elements for the current position
```

```
        for (int j = 0; j < N; j++)
        {
                X[i] = j;
                if (isValid(i))
                {
                        used[j] = true;
                        back(i + 1);
                        used[j] = false; //revert the released column
                }
        }
}
//-------------------------------------
int main()
{
    N = 4;
        X.resize(N);
        used.resize(N, false);
        back(0);
}
```

Output

```
(0, 1) (1, 3) (2, 0) (3, 2)
(0, 2) (1, 0) (2, 3) (3, 1)
```

Sample Problem 2: Maze (Backtrancking in 2D)

Given a maze, NxN matrix. A rat has to find a path from source to destination. maze[0][0] (left top corner)is the source and maze[N-1][N-1](right bottom corner) is destination. There are few cells which are blocked, means rat cannot enter into those cells. Rat can move in four directions: left, right, up and down.

Approach:

- Create a solution matrix of the same structure as maze.
- Whenever rat moves to cell in a maze, mark that particular cell in solution matrix.
- At the end print the solution matrix, follow that 1's from the top left corner, it will be that path for the rat.

Algorithm:

1. If rat reaches the destination
    - print the solution matrix.
    - Else
        - Mark the current cell in solution matrix as 1
        - If previous step is not in vertical direction (upwards) then move forward in the vertical direction(downwards) and recursively check if this movement leads to solution.
        - If movement in above step doesn't lead to the solution and If previous step is not in horizontal direction (towards left) then move forward in the horizontal direction(towards right) and recursively check if this movement leads to solution.
2. If movement in above step doesn't lead to the solution and If previous step is not in vertical direction (downwards) then move forward in the horizontal direction(upwards) and recursively check if this movement leads to solution.

3. If movement in above step doesn’t lead to the solution and If previous step is not in horizontal direction (towards right) then move forward in the horizontal direction(towards left) and recursively check if this movement leads to solution.
4. If none of the above solution works then BACKTRACK and mark the current cell as 0.

Code:

```
// Maze.cpp : Backtracking in 2D
#include "pch.h"
#include <iostream>
#include <vector>
#include <iomanip>

using namespace std;

int N;
vector<vector<int>> maze;
vector<int> dirX  { -1, 0, 1, 0 };
vector<int> dirY  { 0, 1, 0, -1 };
//-----------------------------------------
void printOneSolution()
{
	for (int i = 0; i < N; i++)
	{
		for (int j = 0; j < N; j++)
			cout <<setw(3)<<maze[i][j];
		cout << endl;
	}
	cout << endl;
}
//-----------------------------------------
bool isValid(int x, int y)
{
	if (x < 0 || x >= N || y < 0 || y >= N)
		return false; //out of borders
	if (maze[x][y] != 1)
		return false; //wall or already visited
	return true;
}
//-----------------------------------------
void back(int x, int y)
{
	if (x == N - 1 && y == N - 1)
	{
		printOneSolution();
		return;
	}

	//try to extend to the four directions
	for (int j = 0; j < 4; j++)
	{
		int xx = x + dirX[j];
		int yy = y + dirY[j];
		if (isValid(xx,yy))
		{
			maze[xx][yy] = maze[x][y] + 1;
			back(xx,yy);
			maze[xx][yy] = 1;
		}
	}
}
//-----------------------------------------
int main()
{
	N = 5;
	//0 is a wall
```

```
        maze = {        { 1, 0, 1, 1, 1 },
                        { 1, 1, 1, 0, 1 },
                        { 1, 0, 0, 1, 1 },
                        { 1, 1, 1, 1, 0 },
                        { 0, 0, 0, 1, 1 }
                };
        //starting position
        maze[0][0] = 2;
        back(0, 0);
}
```

Output:

```
  2   0   6   7   8
  3   4   5   0   9
  1   0   0  11  10
  1   1   1  12   0
  0   0   0  13  14

  2   0   1   1   1
  3   1   1   0   1
  4   0   0   1   1
  5   6   7   8   0
  0   0   0   9  10
```

Practice Problems

1. **Permutations:** Printing all permutations of an array. (Manual implementation of the STL next_permuation method.)
2. **Knight's Tour Problem:** It is a sequence of moves of a knight on a chessboard such that the knight visits every square only once.
3. **Subset Sum Problem:** Subset sum problem is to find subset of elements that are selected from a given set whose sum adds up to a given number K. Make a program to calculate number of such subsets.
4. **4-Coloring Problem:** Given an undirected graph determine in how many different ways graph can be colored with at most 4 colors such that no two adjacent vertices of the graph are colored with the same color.
5. **Longest Simple Path Problem:** Find the longest simple path between two vertices in an undirected weighted graph. Graph may have cycles and wegih values can be negative or positive.

Useful Links and References

Programming Challenges (Stevan S. Skiena, Miguel A. Revilla)
http://jeffe.cs.illinois.edu/teaching/algorithms/book/02-backtracking.pd
https://leetcode.com/problems/n-queens/
https://algorithms.tutorialhorizon.com/backtracking-rat-in-a-maze-puzzle/