YILDIZ TEKNİK ÜNİVERSİTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

YAPISAL PROGRAMLAMAYA GİRİŞ

Backtracking (Geri İzleme) Algoritması

Burak Boz

18011706

Video Anlatım Linki

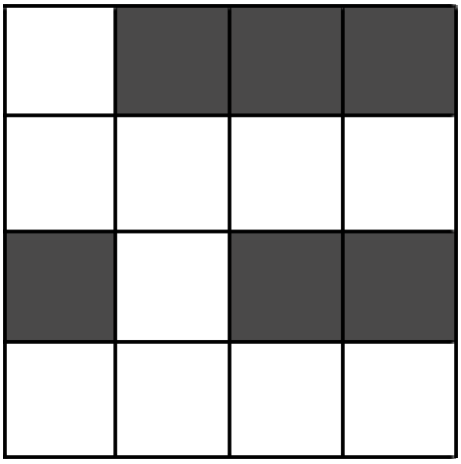
https://www.youtube.com/watch?v=UKnNYDxoRjU

Algoritmanın Tarifi

Backtracking algoritması, bilgisayar bilimlerinde bir değerin aranması ya da bir hedefe ulaşmak için kullanılan algoritmalardandır. Hedefe giderken bulunan çeşitli yollardan doğru olanını tespit etmek için kullanılır. Temel olarak deneme yanılma prensibiyle çalışır. Belirlenen yönde ilerlemeye devam ederek sonuca ulaşmaya çalışır. Yanlış yolda olduğunu fark ettiği anda en son yaşamış olduğu yol ayrımına kadar geri dönerek elde kalan diğer yolları dener ve sonuca ulaşmaya çalışır. Amaç doğru yolu bulmaktır. Birden fazla doğru yol olabilir. Bu algoritmanın bulduğu yolun en kısa yol olma zorunluluğu yoktur. Doğru yolu bulmaya odaklanmıştır.

Çalışma Prensibi

Backtracking algoritmasının farklı kullanım alanları mevcuttur. Rahat anlaşılabilmesi için labirent örneği kullanacağız.



Elimizde bu şekilde bir 4x4 boyutlarında bir labirent bulunsun. Siyah alanlar duvarları, beyaz alanlar ise yolları temsil etsin. Sol üst köşe giriş ve sağ alt köşe de çıkış olsun.

Sol üst köşeden yani 0,0 koordinatından bırakılan bir farenin çıkış yolunu Backtracking algoritması ile bulmaya çalıştığını varsayalım. Fare başlangıç noktasından bırakıldıktan sonra 1,0 konumuna gelecek ve ardından 1,1 konumuna gelerek yol ayrımıyla karşılaşacaktır. Burada eğer x yönüne doğru devam etmeye karar verirse 2 adım sonra yani 1,3 koordinatında gittikten sonra yolu kalmayacak ve geri dönmek zorunda kalacaktır. 1,1 konumuna kadar geri gelecek ve diğer yolu seçerek yoluna devam ederek 2,1 koordinatına varacaktır. Buradan aşağıya inerek 3,1 koordinatına varacak ve x yönünde devam ederek çıkışa ulaşacaktır.

Backtracking algoritması bu prensiple çalışmaktadır. Belirlenen direktifler ışığında deneyerek sonuca ulaşmaya çalışır. Eğer belli bir noktada sıkışırsa en son yaşamış olduğu yol ayrımına kadar geri döner ve diğer yolu dener.

Uygulama Alanları

Backtracking algoritmasının farklı uygulama alanları vardır. Labirent örneği bunlardan bir tanesidir.

Satranç oyununun yapay zekalarında da kullanılabilir. Oynama sırası yapay zekaya geçtiğinde belirli bir taş ile hamleleri denemeye başlar. Eğer ulaştığı son noktada yapılacak hamle işimize yaramıyor veya taş kaybetmemizle sonuçlanıyorsa geri döner ve yol ayrımlarından farklı hamleler deneyerek bulduğu en mantıklı hamleyi oynar.

Sudoku çözümlerinde de Backtracking algoritmasından faydalanabiliriz. Belli bir sayıdan başlayarak alanlara sayıları yerleştiririz. Eğer bir yerde sıkışırsak geri döneriz ve yol ayrımında kullandığımız sayıyı değiştiririz.

Karmaşıklık Analizi

Zaman karmaşıklığı	:	$O(2^{(n\hat{}2)})$
Hafıza karmaşıklığı	:	$O(n^2)$

Avantajlar ve Dezavantajlar

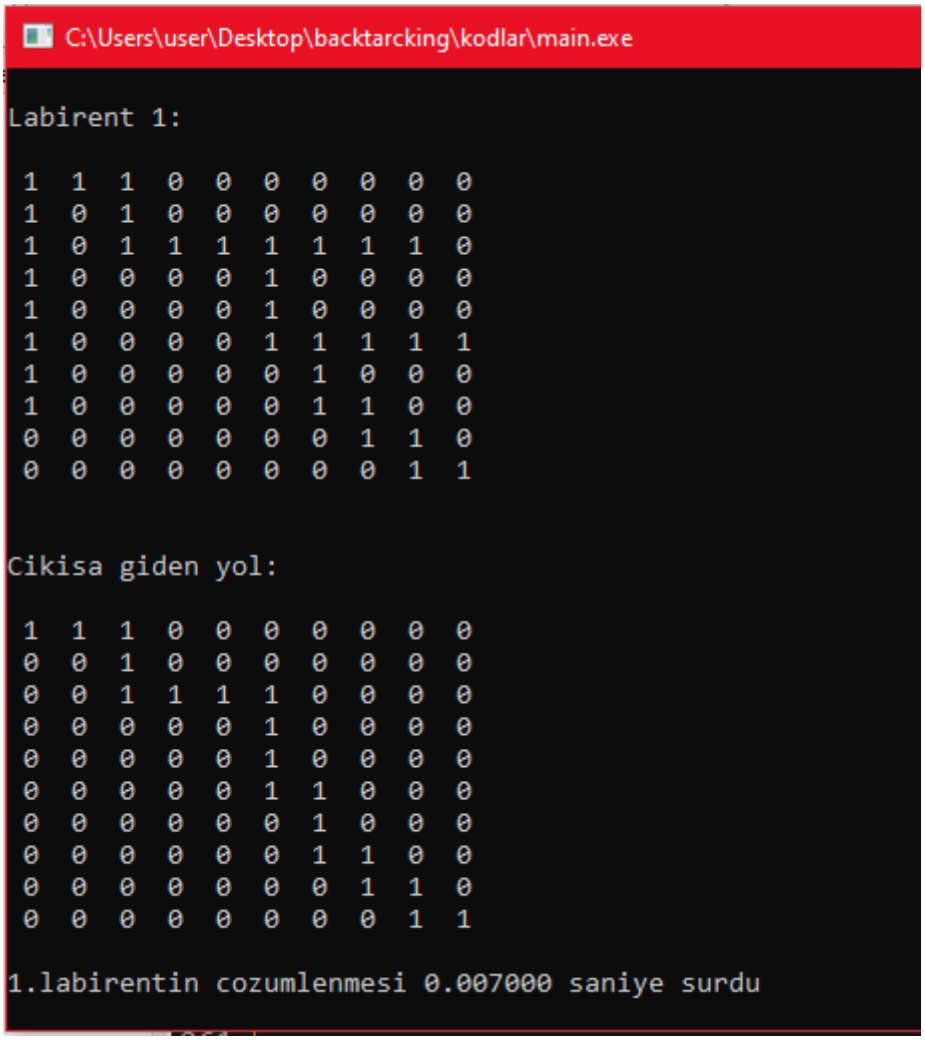
Backtracking algoritmasının temel avantajı doğruluk payıdır. Deneme yanılma yöntemi ile sonuca ulaştığı için doğruluk payı yüksektir. Yüksek hafıza tüketimine ihtiyaç duymaz. Yığın yöntemi kullanılarak gidilen adımlar yığına aktarılabilir ve geri dönerken yığından çekilerek yol ayrımlarına kadar geri gelinebilir. Dolayısıyla hafıza miktarı atılan adımla doğru orantılıdır.

Dezavantajı ise deneme yanılma yöntemi ile çalışmasıdır. Optimum olmayan durumlarda tek bir doğru bulmak için tüm ihtimalleri gezmek gerekebilir. Dolayısıyla zaman bakımından hızlı çalışması gereken durumlarda farklı alternatiflere yönlenmemiz gerekebilir ya da algoritmanın üzerine geliştirmeler eklemek gerekebilir.

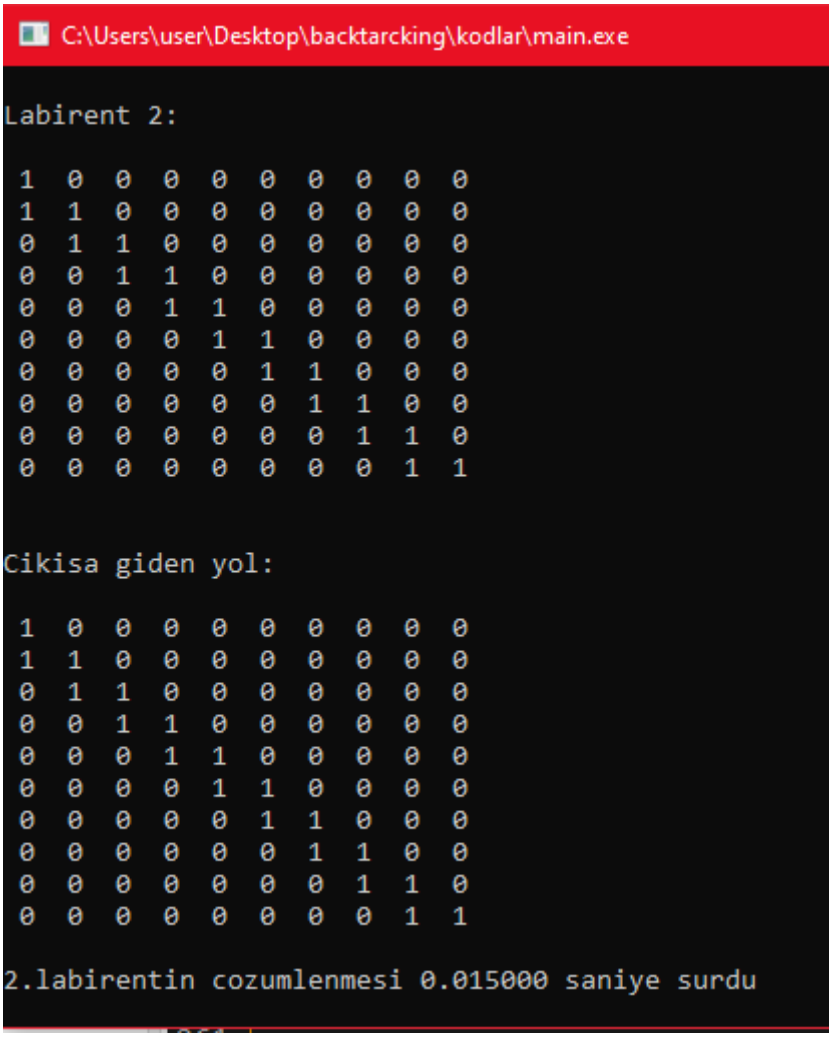
Farklı Girdiler İçin Ekran Çıktıları

Ön bilgi: Labirentlerde 1'ler yolları, 0'lar ise duvarları temsil etmektedir. Sol üst köşe başlangıcı ve sağ alt köşe ise çıkışı temsil etmektedir. Çıktı görsellerinde üstte bulunan labirentin orijinal hali ve altta görünen ise bulunan çıkış yoldur. Görsellerin alt kısmında labirentin çözülmesinin kaç saniye sürdüğü yazmaktadır.

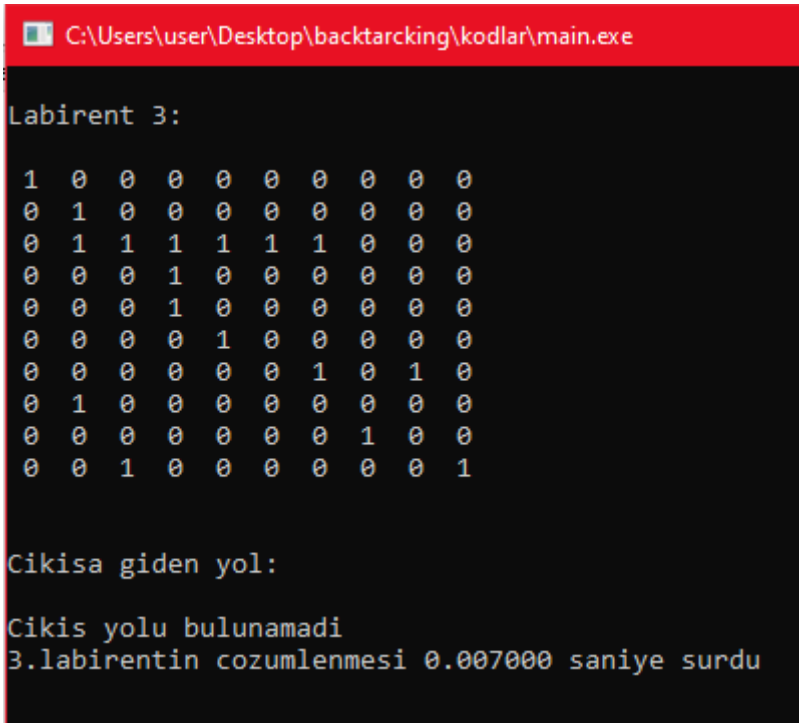
Örnek Labirent 1:



Örnek Labirent 2:

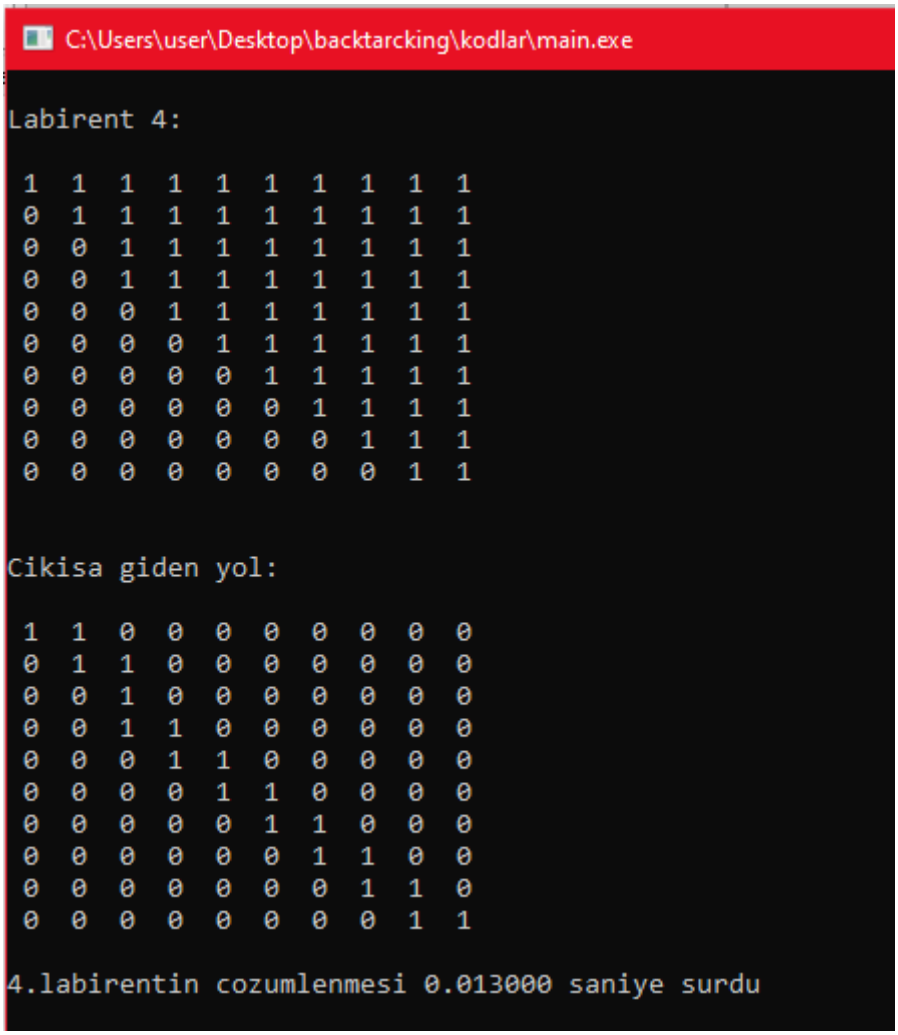


Örnek Labirent 3:

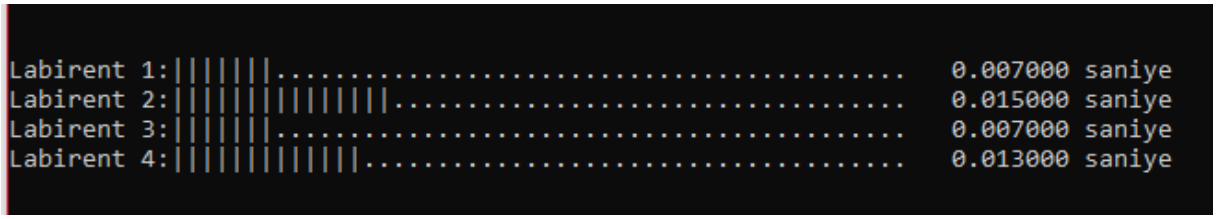


Bu labirentte çıkış yolu bulunamadığı için ekrana çıkış yolu yazdırılamamış ve çıkış yolunun bulunamadığı bilgisi yazdırılmıştır.

Örnek Labirent 4:



Grafik



Yukarıdaki grafikte, belirtilen 4 labirentin işlemlerinin kaç saniye sürdüğü gösterilmektedir. Süre ölçümü için C'nin time kütüphanesi kullanılmıştır. Grafik için verilerin virgülden sonraki 3 basamağı dikkate alınmıştır.

C İmplementasyonu

```
/*
Created By Burak Boz on 08.07.2020
İsim & Soyisim: Burak Boz
Öğrenci Numarası:18011706
Bölüm: Bilgisayar Mühendisliği
Ders Yürütücüsü: Hafiza İrem Türkmen
*/
#include <stdio.h>
#include <time.h>
#include <stdbool.h>

// Labirent boyutu
#define N 10


bool solveMazeUtil(
   int maze[N][N], int x,
   int y, int sol[N][N]);

//Bu fonksiyon matrisleri ekrana yazdırmak için kullanılmıştır.
void printSolution(int sol[N][N])
```

```
{
            int i=0,j=0;
   for (i = 0; i < N; i++) {
      for (j = 0; j < N; j++)
         printf(" %d ", sol[i][j]);
      printf("\n");
   }
}

//Bu fonksiyon matristeki index değerlerinin geçerli olup olmadığını kontrol eder..
bool isSafe(int maze[N][N], int x, int y)
{
   //x ve y değerleri matris sınırları dışındaysa false döndürür.
   if (
      x >= 0 && x < N && y >= 0
      && y < N && maze[x][y] == 1)
      return true;

   return false;
}

/*
Bu fonksiyon labirenti çözmek için backtracking algoritmasını kullanır.
Öncelikle solveMazeUtil() fonksiyonu kullanılır fakat false döndürülürse çözüm yok demektir.
Eğer false dönmez ise mevcut index 1 olarak sol matrisine yazdırılır.
*/
bool solveMaze(int maze[N][N])
{
   int sol[N][N] = {
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
```

```
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 0, 0},
};

   if (solveMazeUtil(
         maze, 0, 0, sol)
      == false) {
      printf("Cikis yolu bulunamadi");
      return false;
   }

   printSolution(sol);
   return true;
}

//Labirenti çözmek için kullanılan özyinelemeli fonksiyon
bool solveMazeUtil(
   int maze[N][N], int x,
   int y, int sol[N][N])
{
   //Eğer x ve y indeksleri geçerli ise true döndürür.
   if (
      x == N - 1 && y == N - 1
      && maze[x][y] == 1) {
      sol[x][y] = 1;
      return true;
   }

   //x ve y değerlerinin geçerli olup olmadığını kontrol eder.
```

```
    if (isSafe(maze, x, y) == true) {
        //Eğer değerler geçerliyse matrise 1 değerini yazar.
        sol[x][y] = 1;

        //x yönünde ileri gider.
        if (solveMazeUtil(
                maze, x + 1, y, sol)
            == true)
            return true;

        //Eğer x yönüne gidemiyorsa y yönünde ileri gider.
        if (solveMazeUtil(
                maze, x, y + 1, sol)
            == true)
            return true;

        //Eğer yukarıdaki iki yolda tıkalıysa geri dönmesi gerektiğini anlar.
        sol[x][y] = 0;
        return false;
    }

    return false;
}

// Ana fonksiyon
int main()
{
            double t1,t2,t3,t4;

            clock_t start_t, end_t;
    start_t = clock();

            int maze[N][N] = {
```

```
{ 1, 1, 1, 0, 0, 0, 0, 0, 0, 0},

{ 1, 0, 1, 0, 0, 0, 0, 0, 0, 0},

{ 1, 0, 1, 1, 1, 1, 1, 1, 1, 0},

{ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0},

{ 1, 0, 0, 0, 0, 1, 0, 0, 0, 0},

{ 1, 0, 0, 0, 0, 1, 1, 1, 1, 1},

{ 1, 0, 0, 0, 0, 0, 1, 0, 0, 0},

{ 1, 0, 0, 0, 0, 0, 1, 1, 0, 0},

{ 0, 0, 0, 0, 0, 0, 0, 1, 1, 0},

{ 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},

};


        printf("*******Burak Boz - Backtracking Algoritmasi*******\n\n");

        printf("** Labirentin sol ust kosesi baslangici ve sag alt kosesi cikisi temsil etmektedir.\n** 1 ler yol, 0
lar ise duvarlari temsil etmektedir.\n\n");

        printf("Labirent 1:\n\n");

        printSolution(maze);

        printf("\n\nCikisa giden yol:\n\n");

   solveMaze(maze);


   end_t = clock();

   t1=(double)(end_t - start_t) / CLOCKS_PER_SEC;

   printf("\n1.labirentin cozumlenmesi %f saniye surdu\n", t1);



   printf("\n--------------------------------\n\n");



   int maze2[N][N] = {

{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0},

{ 1, 1, 0, 0, 0, 0, 0, 0, 0, 0},

{ 0, 1, 1, 0, 0, 0, 0, 0, 0, 0},

{ 0, 0, 1, 1, 0, 0, 0, 0, 0, 0},
```

```
{ 0, 0, 0, 1, 1, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 1, 1, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 1, 1, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 1, 1, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 1, 1, 0},
{ 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
};

   start_t = clock();

   printf("Labirent 2:\n\n");
            printSolution(maze2);
            printf("\n\nCikisa giden yol:\n\n");
   solveMaze(maze2);

   end_t = clock();
   t2=(double)(end_t - start_t) / CLOCKS_PER_SEC;
   printf("\n2.labirentin cozumlenmesi %f saniye surdu\n", t2);

   printf("\n--------------------------------\n\n");



   int maze3[N][N] ={
{ 1, 0, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 1, 1, 1, 1, 1, 1, 0, 0, 0},
{ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 1, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 1, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 1, 0, 1, 0},
{ 0, 1, 0, 0, 0, 0, 0, 0, 0, 0},
{ 0, 0, 0, 0, 0, 0, 0, 1, 0, 0},
```

```
{ 0, 0, 1, 0, 0, 0, 0, 0, 0, 1},
};

    start_t = clock();

    printf("Labirent 3:\n\n");
            printSolution(maze3);
            printf("\n\nCikisa giden yol:\n\n");
    solveMaze(maze3);

    end_t = clock();
    t3=(double)(end_t - start_t) / CLOCKS_PER_SEC;
    printf("\n3.labirentin cozumlenmesi %f saniye surdu\n", t3);



    printf("\n--------------------------------\n\n");


    int maze4[N][N] ={
{ 1, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{ 0, 1, 1, 1, 1, 1, 1, 1, 1, 1},
{ 0, 0, 1, 1, 1, 1, 1, 1, 1, 1},
{ 0, 0, 1, 1, 1, 1, 1, 1, 1, 1},
{ 0, 0, 0, 1, 1, 1, 1, 1, 1, 1},
{ 0, 0, 0, 0, 1, 1, 1, 1, 1, 1},
{ 0, 0, 0, 0, 0, 1, 1, 1, 1, 1},
{ 0, 0, 0, 0, 0, 0, 1, 1, 1, 1},
{ 0, 0, 0, 0, 0, 0, 0, 1, 1, 1},
{ 0, 0, 0, 0, 0, 0, 0, 0, 1, 1},
};

    start_t = clock();
```

```
printf("Labirent 4:\n\n");
	printSolution(maze4);
	printf("\n\nCikisa giden yol:\n\n");
solveMaze(maze4);

end_t = clock();
t4=(double)(end_t - start_t) / CLOCKS_PER_SEC;
printf("\n4.labirentin cozumlenmesi %f saniye surdu\n\n\n", t4);


int time1,time2,time3,time4;//Süreleri grafiğe dökebilmek için iteger değerlere çevireceğiz.
                                    //Virgülden sonraki 3 basamak dikkate alınacaktır. 1000 ile çarpacağız.

time1=(int)(t1*1000);
time2=(int)(t2*1000);
time3=(int)(t3*1000);
time4=(int)(t4*1000);
int i=0;

printf("Labirent 1:");
	for(i=0;i<50;i++)
	{
		if(i<time1)
		{
			printf("|");
		}
		else{
			printf(".");
		}
	}
	printf("\t%f saniye \n",t1);
```

```
printf("Labirent 2:");
for(i=0;i<50;i++)
{
	if(i<time2)
	{
		printf("|");
	}
	else{
		printf(".");
	}
}
printf("\t%f saniye \n",t2);
printf("Labirent 3:");
for(i=0;i<50;i++)
{
	if(i<time3)
	{
		printf("|");
	}
	else{
		printf(".");
	}
}
printf("\t%f saniye \n",t3);

printf("Labirent 4:");
for(i=0;i<50;i++)
{
	if(i<time4)
	{
		printf("|");
	}
	else{
```

```
                    printf(".");
                }
            }
            printf("\t%f saniye \n",t4);
}
```

Kaynakça

- Wikipedia
  https://en.wikipedia.org/wiki/Backtracking

- GeeksForGeeks
  https://www.geeksforgeeks.org/backtracking-algorithms/

- Şadi Evren Şeker
  http://bilgisayarkavramlari.sadievrenseker.com/2009/11/01/geri-izleme-algoritmasi-backtracking-algorithm/