

Implementation of Backtracking Algorithm in Hamiltonian Cycle

Octavianus Marcel Harjono / 13513056

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia

13513056@std.stei.itb.ac.id

Abstract—Backtracking is a general algorithm to find solutions in common problem. The backtracking algorithm is based on Depth-First Search Algorithm, but it is more efficient because it has bounding function in it. Backtracking algorithm is commonly used in games such as tic-tac-toe solver, sudoku solver, and many more. This paper will explain how to find Hamiltonian Circuit from a graph using backtracking algorithm. Hamiltonian circuit is a graph cycle that has a closed loop which path visits each node/vertex exactly once.

Index Terms—Backtracking Algorithm, Hamiltonian Circuit, Hamiltonian Cycle, Graph, DFS-Based Algorithm

I. INTRODUCTION

The Icosian game, introduced by Sir William Rowan Hamilton who was an Irish mathematician, is known as Hamiltonian Circuit (HC) problem. Hamiltonian circuit, also called Hamiltonian cycle, is a graph cycle through a graph that visits each node exactly once except for the starting node (which also the ending node) is twice. A graph is said to be Hamiltonian if it contains Hamiltonian Circuit, otherwise the graph is nonhamiltonian. The Icosian game itself uses graph like Dodecahedron which is Platonic solid and has Hamiltonian Circuit in it.

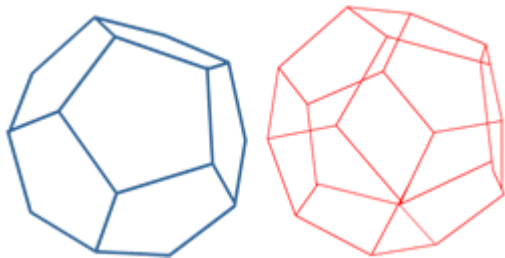


Figure 1. Dodecahedron Example
(<http://nrich.maths.org/2320>)

II. FUNDAMENTAL THEORY

A. Depth-First Search Algorithm

Depth-First Search algorithm (DFS algorithm) is one of common strategy to find solution. It either uses a dynamic tree which is created along the searching or uses a given tree. DFS algorithm, like Breadth-First Search algorithm (BFS algorithm) has several components, they are:

1. Problem state: nodes within the dynamic tree that

fulfill the constraints

2. Solution state: one or more states that declare problem state.
3. Goal state: solution state which is a leaf node.
4. Solution space: set of all solution states.
5. State space: all nodes in the dynamic tree

B. Backtracking Algorithm

Backtracking algorithm is a DFS-based (Depth-First Search based) algorithm. While DFS algorithm will try to find solution from root to each leaf, backtracking algorithm will just find solution from root to a certain depth depending on its bounding function.

There are several important components on backtracking algorithm:

1. Solutions

Solution is declared by an n-tuple vector:

$$X = (x_1, x_2, \dots, x_n), x_i \in S_i$$

It is possible that $S_1 = S_2 = \dots = S_n$

2. Generating function for x_k

Declared as a predicate: $T(k)$

$T(k)$ generates value for x_k , which is a component from vector of solutions.

3. Bounding function

Declared as a predicate: $B(x_1, x_2, \dots, x_k)$

B is true if (x_1, x_2, \dots, x_k) leads to a solution.

If it is true, then generating function for x_{k+1} continues, if it is false, then (x_1, x_2, \dots, x_k) is thrown away from the solutions.

There are several principles in finding solutions with backtracking algorithm, they are:

1. Solutions are searched by making a path from root to leaves. This is from DFS algorithm. The nodes that have been created are called 'live node'. Live nodes that are being expanded are called 'expand-node'.
2. Every time expand-node is expanded, path that has been built is longer. If the path does not lead to the solution, the path is 'killed' and becomes a dead node. Function used for killing this expand-node is bounding function. Dead node will never be expanded again.
3. If the process of making path ends with dead node, then the searching continues by evoking another

child node. If there is no more child node, then it will evoke the nearest parent node alive. This node will be the new expand-node.

- The searching ends if we have found the solution or if there is no more live node for backtracking.

C. Graph

Graph is used to represent discrete objects and the relation within the objects. Graf is a set of vertices that are not empty and a set of edges which connect a pair of vertices. Graph's notation is $G = (V, E)$ where G is Graph, V is set of vertices v_1, v_2, \dots, v_n and E is set of edges e_1, e_2, \dots, e_n .

Graph's terminology:

- Adjacency**
Two vertices are adjacent if both vertices are directly connected.
- Incidence**
For any edge $e = (v_j, v_k)$, e is said incident with vertex v_j , or e is incident with vertex v_k .
- Isolated vertex**
Isolated vertex is a vertex that does not have edge that is incident with it.
- Empty graph**
Empty graph is a graph which set of edges is an empty set.
- Degree**
A degree of a vertex is the number of edges which are incident with the vertex.
- Path**
A path with length n from starting vertex v_0 to v_n is a line which forms 'vertex-edge-vertex' pattern. There are Euler path and Hamilton path.
- Cycle**
Cycle or circuit is a path which starts and ends at the same vertex. There are Euler circuit and Hamilton circuit.
- Connected**
Two vertices are connected if there exists a path between those vertices.
- Subgraph**
If we have a graph $G = (V, E)$, then graph $G_1 = (V_1, E_1)$ is a subgraph of G if V_1 is subset of V , and E_1 is a subset of E .
- Spanning Subgraph**
Subgraph G_1 is called spanning subgraph if G_1 has all vertices from G .
- Cut-set**
Cut-set is a set of edges that if the edge is deleted, it will cause G not connected Cut-set always yields two components.
- Weighted graph**
Weighted graph is a graph which every edge of it is given a weight or value.

D. Hamiltonian cycle

On section C (Graph), there are several terminologies about graph. One of it is cycle (point 7). Hamiltonian cycle

is a cycle or circuit which goes through every vertex exactly once, except for the starting vertex (which also the ending vertex) which is twice. There may be more than one Hamilton circuit for a graph, and then we often wish to solve for the shortest such path. This is often referred to as traveling salesman problem (TSP) but the graph is a complete weighted graph. Every complete graph ($n > 2$) has a Hamilton circuit. (<http://www.pballew.net/graphs.html>)

Dirac's Theorem: If each vertex of a connected graph with n vertices (where $n > 3$) is adjacent to at least $n/2$ vertices, then the graph has a Hamilton circuit.

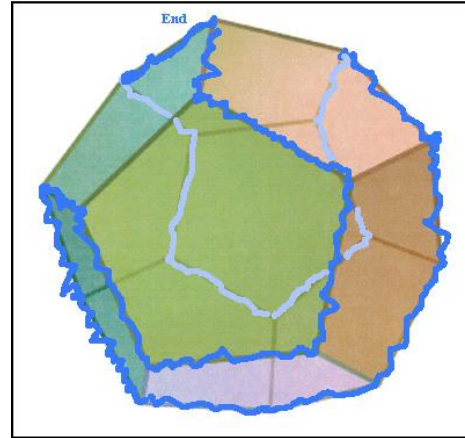


Figure 2. Dodecahedron with Hamiltonian Cycle in it. (<http://curvebank.calstatela.edu/hamiltoncircuit/hamiltoncircuit.htm>)

III. ANALYSIS AND IMPLEMENTATION

A. Backtracking Algorithm Implementation

Backtracking Algorithm (recursive):

```

procedure backtracking(input k: integer)
{Finding all solutions with backtracking algorithm;
iterative scheme
Input: k, the index of solutions vector, x[k]
Output: Solution x = (x[1], x[2], ..., x[n])
}
ALGORITHM
for each x[k] which has not been tried so that
x[k] ← T(k) and B(x[1], x[2], ..., x[k]) do
if(x[1], x[2], ..., x[k] is path from root to leaf)
then
PrintSolution(x)
endif
backtracking(x)
endfor

```

Backtracking Algorithm (iterative):

```

procedure backtracking(input n: integer)
{Finding all solutions with backtracking algorithm;
iterative scheme
Input: n, the length of solutions vector
Output: Solution x = (x[1], x[2], ..., x[n])
}

```

DICTIONARY

k: integer

ALGORITHM

```

k ← 1
while k>0 do
  if (x[k] has not been tried so that x[k] ← T(k) and
  B(x[1], x[2], ..., x[k])) then
    if(x[1], x[2], ..., x[k] is path from root to leaf)
    then
      PrintSolution(x)
    Endif
    k ← k + 1
  else
    k ← k - 1
  endif
endwhile
{k = 0 }

```

B. Illustrative Example

In this paper we will use backtracking algorithm to find solutions to Hamiltonian Circuits Problem. The idea to find the solution of it is like the idea of backtracking algorithm itself. Let us say we have a connected graph with n = 8 like figure 3 below.

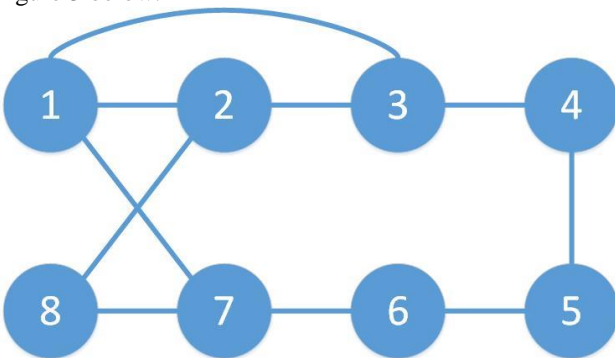


Figure 3. Example of Graph with Eight Vertices

First we have to choose a root node, in Hamiltonian Circuits is the starting vertex. Then we have to make the bounding function. In Hamiltonian cycle, the bounding functions is: 1. Every node selected has not been selected before, 2. Every node selected must have edge connecting it to previous node (these vertices have to be adjacent), 3. Last node selected has to be the first node selected.

After making the bounding function, we have to explore the root node and see whether it has edge along the remaining nodes. In this case if we choose vertex 1 as the starting vertex (which also the ending vertex), then we have vertex 2 to vertex 8 as the remaining nodes. We have to check whether vertex 1 has edge connecting to each vertex 2 until vertex 8. If it does not have the connecting edge, then it becomes dead node, so we do not have to expand the node.

In this case, vertex 1 is only adjacent with vertex 2 and

vertex 3, so the rest of it become dead node.

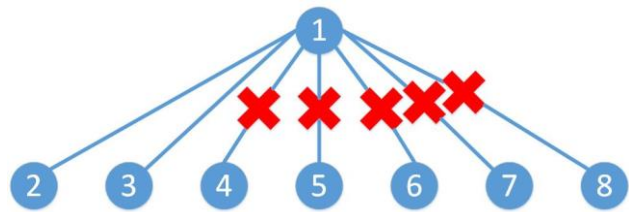


Figure 4. State Space Tree

Then we only can choose node 2 or node 3.

Doing it continuously, and if the expand-node has a child node which is the starting vertex but it has not gone through every other vertex, it becomes dead node too because it is not a Hamiltonian circuit.

If it has reach the leaf and it is still a dead node, then we have to evoke the nearest child node, and if it has no alive child node, we have to evoke the nearest parent node.

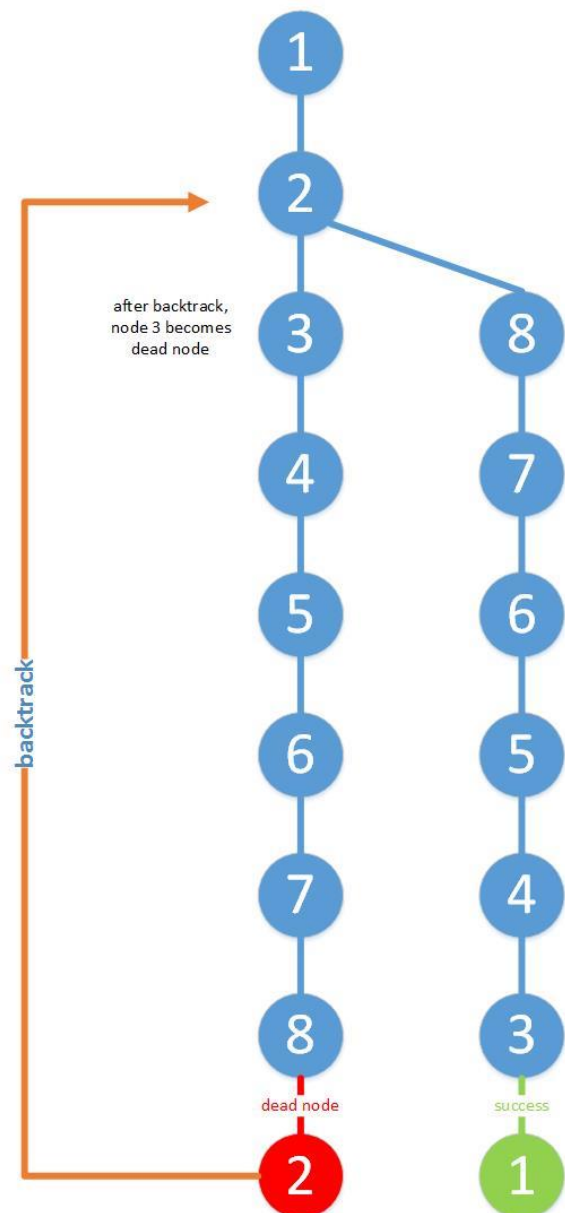


Figure 5. Final State Space Tree

In every step of expanding the node, it should be like at

Figure 4, every vertex that has no edge connecting them is thrown away from solutions vector.

Since it has found the solution we are looking for, so it just returns the solutions vector $\langle 1,2,8,7,6,5,4,3,1 \rangle$

In this example, it just looks like DFS because there are few edges in most vertices and we cannot check if there exists Hamiltonian cycle using Dirac's Theorem because not all vertex has at least four other vertices adjacent to it.

In this case, suppose that it is not the solution, then we have to go backtracking to the root (vertex 1), make the node to vertex 2 a dead node, and then expand the node to vertex 3, and so on.

C. Analysis on The Algorithm

The difference between DFS and Backtracking algorithm on finding solutions for Hamiltonian Circuit is the restriction (bounding function). The backtracking algorithm is said to be more efficient than DFS because in every step in backtracking algorithm, it has thrown away all the nodes that do not lead to the solutions.

V. CONCLUSION

The backtracking algorithm explained in this paper is only a pseudo code but it can be implemented and it can produce the right solutions to Hamiltonian Circuit problem. Although backtracking algorithm is based on Depth-First Search algorithm, his algorithm is more efficient than DFS algorithm because it has bounding function in it which forbids the program to continue expanding the node that does not lead to solution.

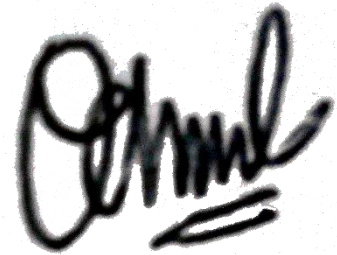
REFERENCES

- [1] <http://mathworld.wolfram.com/HamiltonianCycle.html>
accessed: 4 May 2014, 23:30
- [2] Munir, Rinaldi. Diktat Kuliah IF2211 Strategi Algoritma. Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, 2008, page 167 -185.
- [3] <http://nrch.maths.org/2320>
accessed: 4 May 2014, 23:51
- [4] <http://curvebank.calstatela.edu/hamiltoncircuit/hamiltoncircuit.htm>
accessed: 4 May 2014, 23:56
- [5] <http://www.pballew.net/graphs.html>
accessed: 4 May 2014, 23:59

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 4 May 2015



Octavianus Marcel Harjono / 13513056