ISSN 1870-4069

Significance of Algorithmic Approaches in Graph Theoretical Concepts and Searching Techniques

Muhammad Ateeb Ather

Department of Computer Sciences, Bahria University, Lahore, Pakistan

03-134211-022@student.bahria.edu.pk

Abstract. Perhaps no field or subfield of computer sciences is more intriguing and research about than algorithms, their significance and their application throughout different disciplines of sciences. Algorithms are fundamental foundations for many domains including modelling, testing, qualifying and employing in complex systems developed for different purposes. One of the biggest breakthrough in the history of science, Turing Machine also uses this very technique. Other disciplines also use different algorithms vastly to explain, express and implement theories and propositions. One such discipline is graph theory which deals with modeling, representing and expressing different propositions and empirical theories. It comes as no surprise that it also employs the algorithm technology. Having said that, it has not gained the emergence within research.

Keywords: Algorithms, graph theory, graph representation, shortest path problem, graph coloring.

1 Introduction

As the computers becomes more powerful and faster, on the same way the importance of algorithms increases. In computer science applications, development of algorithms plays a vital role. To build a large program with an appropriate time and space consumption, it is important to have well-organized solutions to the problem parts. The most important part of algorithm is that how much it is fast. Computer scientists usually talk about the runtime relative to the size of the input. However, many complex algorithms execution time can differ due to the number of factors other than the size of the input.

Algorithms plays an important role in our lives from various aspects in the field of computer science. Like in flying, making money, searching, weather forecasting, find shortest path to travel from one location to another and much more. Now everyone uses a computer so every time, when we uses the search engine and hit the search button for

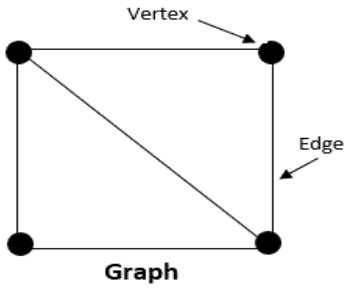


Fig. 1. Representation of Graph.

searching social networks and websites like google, twitter, Facebook etc. it provides result in a fraction of second. This is only possible because of algorithms. They have changed our lives by sorting through the vastness of the internet and giving us relevant, immediate results**.**

As it is known as the computer science field and knowledge is so vast that it comprises of many different sciences, graph theory is also a branch of computer science. Particularly in research domains of computer science, graph theoretical concepts are highly applied by computer science applications like image capturing, clustering and data mining etc. For example, in terms of trees by using vertices and edges data structures could be developed.

Using graph concepts, demonstration of network models can be implemented. Also routes and tracks in graph theoretical concepts are used in numerous applications such as resource networking, designing of database concepts etc. [1].

In computer applications, designing of graph algorithms is an essential part in graph theory. In the form of graph numerous algorithms have been designed to solve problems. These algorithms are being used in solving graph concepts and hence in solving the corresponding computer science application problems.

Searching algorithms(DFS, BFS) Shortest path algo's, Algorithms used to find cycles (rotations) in a graph, Algo's to find adjacency matrices and so on.

In 1736, Graph theory originated from Euler "Konigsberg bridge" problem. Euler's key insight was that by using simple mathematical structure called *graph*, the islands and bridges could be demonstrated. In CS (computer science) and mathematics, graph theory is expressed in terms of the study of graphs. To model pairwise associations between objects (entities), graphs are used as a mathematical structures.

Most of the people use the word "graph" roughly, you can't give any grantee what they are talking about and what type of graph. Because there are different types of graphs use in mathematics, computer science or any other field, with its own description. Section 2 describes the basic concepts of graphs and its types. Section 3 emphasis on the history of graphs. Section 4 focuses on the demonstration of graphs using matrices. Segment 5 describes basic algorithms of graphs.

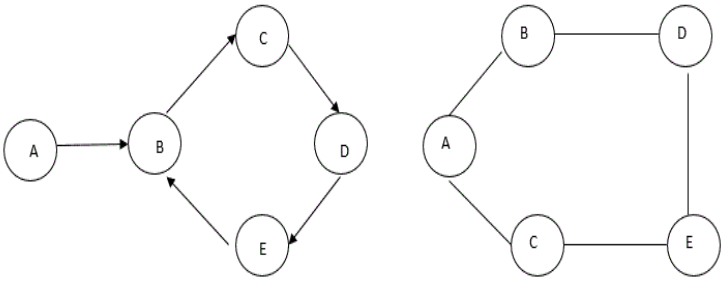


Fig. 2. Directed vs Un-Directed Graph.

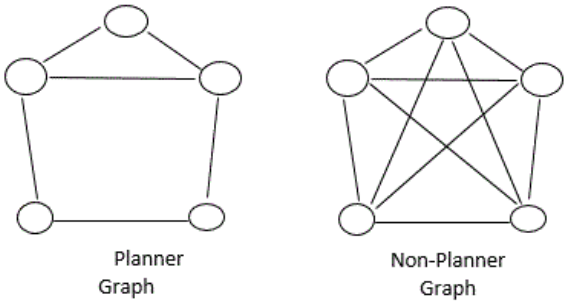


Fig. 3. Planner vs Non-Planner Graph.

2 Graph

According to authors, "Graph is a representative demonstration of a network and its connectivity" [2].

A special diagram called *graph* provides an efficient way of moving among different destinations in a routing problem.

Informally, graph is a map comprising of points called vertices, linked together by lines called edges.

In the above figure Black dots show destinations, while the lines used to join destination points called edge.

Graph is a set of (V, E), components of V are vertices (dots, nodes) and components of E are its edges (arcs, lines). Vertex: V is a joining or ending point. Edge: E is an association between nodes [3].

To analyze a graph it is important to study about degree of a graph. To find the degree of a graph, figure out all of the vertex degrees. The degree of the graph would be its largest vertex degree.

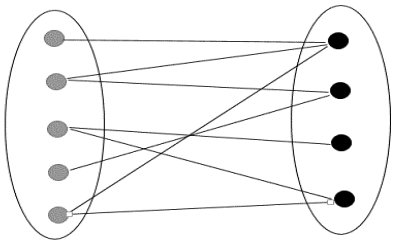


Fig. 4. Bipartite Graph.

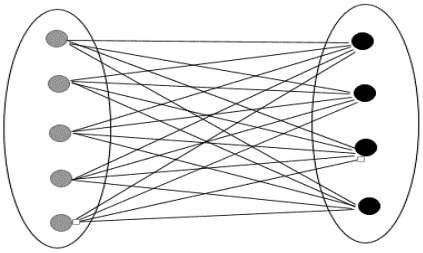
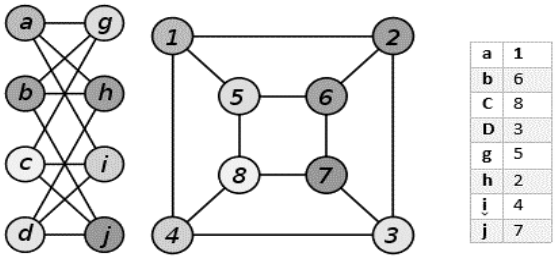


Fig. 5. Complete Bipartite Graph.



a	1
b	6
C	8
D	3
g	5
h	2
i	4
j	7

Fig. 6. Isomorphic Graph.

2.1 Types of Graphs

There are a number of graphs depending on the no of edges, no of vertices, interconnectivity and their overall structure. Here we will describe few important types of graphs.

– Directed and Un-Directed graph: A graph, each edge associates with other vertex must have a direction called DG. A graph contains edges and vertices but edges contains no direction called Un-DG.

– Planner and Non-planner Graphs: a graph will be planner if no link is overlapping with another. Otherwise it will be non-planner graph.

– Simple Graph: a graph does not contains loops and multiple edges.

- Complete Graph: Type of a simple graph in which every couple of dots joined by a link (edge).
- Regular Graph: a type of graph in which all vertices must have the same degree.
- Cyclic Graph: a graph comprises of minimum one cycle. At least one cycle is compulsory.
- Acyclic Graph: graph without cycles. A connected acyclic graph or diagram is known as *tree*, and disconnected acyclic map called *forest*.
- Cycle Graph: circular graph consists of only one cycle.
- Subgraph: graph contains some of the vertices and edges of another graph.
- Bouquet Graph: a graph comprises only one vertex.
- Bipartite Graph: special type of diagram in which set of nodes could be partitioned into two cells (v1, v2) or disjoint sets (two sets are said to be disjoint if have no common element) such that all edges go only between v1 and v2. Edges cannot move from v1 to v1 and from v2 to v2.
- Complete Bipartite Graph: is a special kind of graph, each vertex of one set must be joined to each vertex of another set of the graph.
- Isomorphic graphs: Two graphs that contain same number of edges & same number of vertices joined in the same way called isomorphic.

3 History

In recent past, researchers have focused towards investigating the potential of analyzing graph theory concepts. As discussed earlier graph theory came into existence with the problem of Konigsberg Bridge in 1736. Euler provide a solution of this problem by constructed a structure called Eulerian Graph, so Euler became the father of graph theory. A.F Mobius, introduce concept of bipartite & complete graph in 1840. In 1845, Gustav Kirchhoff implemented the idea of tree (connected graph does not contain cycles [4]) and developed graph theoretical concepts. Thomas found the 4 color problem in 1852. For more than a century, 4 color problem remained unsolved. Many invalid solutions were suggested. Thomas.P.kirkman & William R.Hamilton invented the concept of Hamiltonian graph in 1856. Sylvester introduced the term “graph” in 1878 [5]. Puzzle problem was first introduced in 1913 by H.Dudeney. Another section of graph theory called “Extremal Graph Theory”, were originated in 1941 by Ramsey. After a century, Heinrich provide a solution for 4 color problem by using computers in 1969. This conception of graph coloring is highly applied in the domains like resource allocation and scheduling.

In graph theory various concepts like paths, walks and circuits are used in various applications like TSP, networking etc. This leads to the advancements of new processes, new algorithms, approaches & procedures which are being used in enormous fields or applications [6].

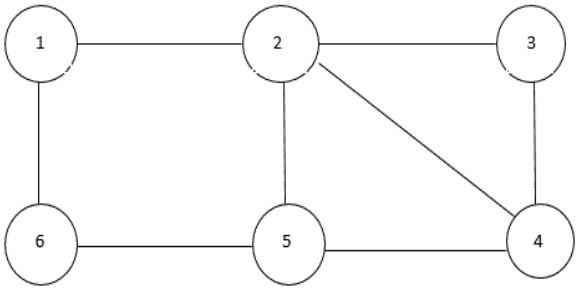


Fig. 7. Representation of Un-Directed Graph.

Node	Neighbors
1	2,6
2	1,3,4,5
3	2,4
4	2,3,5
5	2,4,6
6	1,5

Fig. 8. Adjacency List.

4 Graph Representation

There are different kinds of representations, which can be very beneficial in some situations and inoperable in others. Here we will see three methods to illustrate the graphs.

– Adjacency list.
– Adjacency matrix.
– Edge list.

Graph G= (V, E), might be either undirected or directed. Running time of an algorithm, could be demonstrated in the form of |E| & |V|. We will show the cardinality O (V+E), in asymptotic representation.

4.1 Edge List

One simple way to express a graph is just a list or array of |E| edges, which we call an edge list. To demonstrate an edge, we just have an array of two vertex numbers, or an array of objects comprising the vertex number of vertices that the edges are incident on. In an edge queue L, every edge e is kept as a tuple (vi, vj) in an arranged queue. The edge list of the graph described above is:

$$L = \{ (1,2) , (1,6) , (2,3) , (2,4) , (2,5) , (3,4) , (4,5) , (5,6) \}. \quad (1)$$

0	**1**	**0**	**0**	**0**	**1**
1	0	1	1	1	0
0	1	0	1	0	0
0	1	1	0	1	0
0	1	0	1	0	1
1	0	0	0	1	0

Fig. 9. Adjacency Matrix.

However being a space proficient technique to store the graph, an edge list access time is slower than the adjacency matrix access time (for most edges).

4.2 Adjacency List

It is related to an edge list but difference is, array is associated with a queue. Mass of an array is equal to the no of nodes. An entrance array (group) shows the linked lists of vertices neighboring to the i^{th} node. This demonstration could be used to express a weighted graph. In the nodes of linked list, the edge weights can be stored. Adjacency list demonstration of the graph is.

4.3 Adjacency Matrix

Matrix A, the AM of a graph is V x V, if there exists an edge, each entry is equal to 1 otherwise 0. Note that, graph we are using in the example is undirected, the resulting matrix is symmetric. Also no self-loops exists, each entry in the main diagonal is 0.

5 Graph Algorithms

There are number of algorithms that could be applied to graphs. Many of them are actually applied in real world like shortest path, topological sort, Dijkstra algorithm, DFS, BFS, graph coloring algorithms etc. Here, we will discuss few of them.

5.1 Breadth First Search (BFS)

BFS can be used to discover shortest path from maze problem according to E.F Moore [7]. BFS was first introduced in the late 1950's and exposed independently by C.Y.Lee as a wire routing problem. BFS is a general algorithm for searching or traversing a graph. Before moving to the next level neighbors, BFS discovers the starting node and its neighbor's node first. BFS search start working on the vertex which will be on level zero. There are two phase: In the 1^{st} step, go to all vertices which will be at the distance of one edge away. Go there, paint "visited" the vertices we have been visited adjacent to the starting node, and kept these vertices into the level one. In 2^{nd} step phase, visit

all new vertices that are at the space (distance) of 2 edges away from the starting node. All new vertices that do not previously assigned to a level kept into level two & so on, which are adjacent to level one. When every vertex has been visited, BFS traversal works end.

For traversing graph nodes, this is a very different methodology. The aim of BFS algorithm is to traverse the graph to the root node & to implement this, Queue is used. Now see, how BFS traversal done on the following graph.

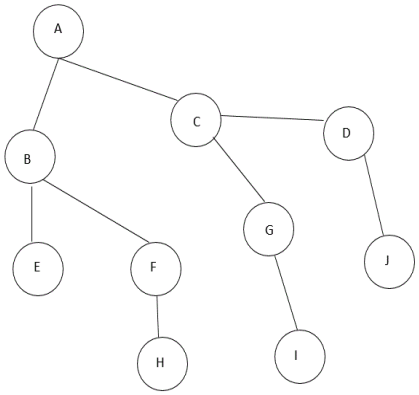


Fig. 10. Simple Graph with 10 vertices.

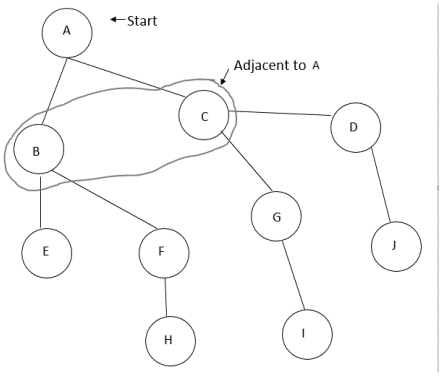


Fig. 11. First BFS Layer.

Pseudocode

BFS (G, s)

1. Q = queue comprising only s
2. while Q not empty
3. v = Q.front(); Q.remove front()
4. for w ∈ G.neighbors(v):
5. if w not seen:
6. mark w seen
7. Q.enqueue(w)

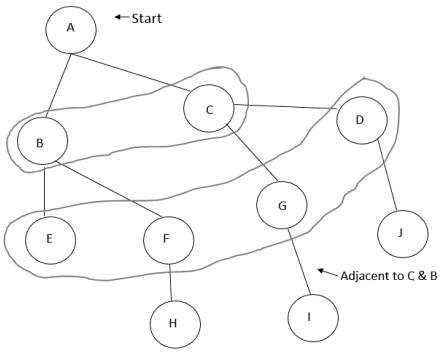


Fig. 12. Second BFS Layer.

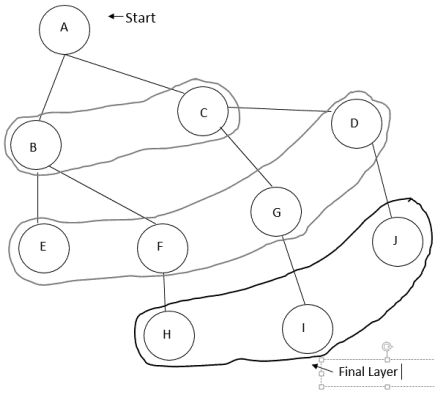


Fig. 13. Final BFS Layer.

For some explanation, this is an undirected graph. So you cannot see arrows. E. g., point from vertex F to vertex H, then the edge b/w vertex F to H can be traversed in either direction. Another point is that a vertex is said to be adjacent to another vertex if an edge joins the 2 vertices directly. When we apply BFS, we can only go from one adjacent vertex to another. You cannot move directly from vertex C to I.

In this algo, we will start from source vertex which is the root of the graph, put the source vertex into the queue. After visiting the root vertex, go to its neighbor nodes, if not visited yet. Queue contains all vertices that have been seen but not yet visited.

Now start BFS if we are at level 1. Put A into queue, unvisited nodes are H, I, G, J, D, F, E, C, B & visited node is A. Dqueue node (A). Now vertex (A) has 2 unvisited vertices C & B.

Now BFS algorithm looks at vertices B & C and mark as visited, so they have been taken placed in queue after vertex A listed its adjacent neighbors. At this stage only vertices A, B & C count as vertices which have been visited. Vertex A has no more unvisited children, Dqueue B. So insert vertices E & F into the queue, also put vertices G & D in the queue. Figure 12 depicts the next layer of BFS with gray wall.

Pseudocode

```
DFS(G, v){
1. mark v visited
2. set color of v to gray
3. for each successor v' of v {
4. if v' not yet visited {
a. DFS(v')}}
5. set color of v to black }
```

Stack Based DFS Pseudocode

```
DFS(G, s):
a. ST = stack containing only s
6. while ST not empty
a. v = ST.pop()
b. if v not visited
c. mark v visited
7. for w ∈ G.neighbors(v): S.push(w)
```

At this level, node E has no more unvisited vertices, Dqueue F, enqueue H into queue. Now we will Dqueue G & D and enqueue their vertices into queue, it includes I & J.

5.1.1 Complexity of BFS

Complexity of BFS could be explained in O (V+E) since each edge & node will be discovered in the worst case. Remind that O (E) may differ b/w O (1) & O (V^2) depending on how scarce the input graph is. If the graph is expressed by an adjacency list it obtains O (V+E) space in memory, while an adjacency matrix depiction occupies O (V^2) [8].

5.1.2 Applications of BFS

In graph theory, BFS could be used to resolve many problems, such as:

- Testing a graph for bipartiteness.
- Copying garbage collection, Cheney's algo.
- In a flow network, compute maximum flow- Ford Fulkerson method
- Finding shortest path b/w 2 nodes etc [9].

BFS can also be used to test bipartiteness, starting search from at any node, during the traverse give differ labels to the vertices visit. Starting node give the name 0, 1 to all its close or neighbors, 0 to those neighbors' neighbors and so on. Graph will not be bipartite, if a node has neighbors with the same label as itself at any phase. If traverse terminates and no situation occur as we said earlier then graph would be bipartite.

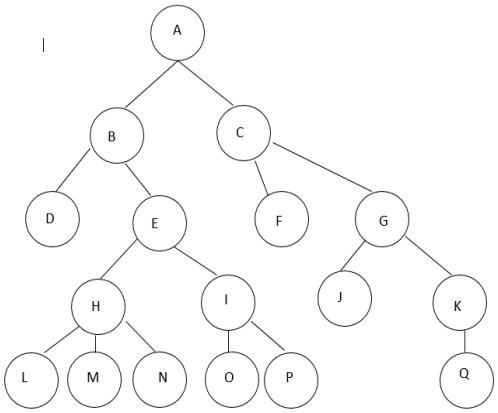


Fig. 14. 1st Depth First Search.

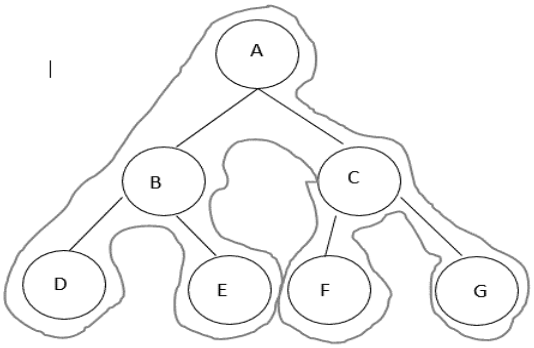


Fig. 15. Depth First Search (DFS).

5.2 Depth First Search (DFS)

French mathematician Charles Pierre Tremaux was discovered DFS in the 19th century as a policy for solving mazes [10-11].

DFS is another approach for traversing a graph. There are a number of algorithms which based on DFS, DFS allows visiting vertices of the graph only. Therefore, to move into the graph theory it's important to learn about the concepts of DFS. Algorithm is quite easy, move until there is a chance to go forward, otherwise go back.

In DFS, every vertex has 3 possible colors, such as:

Gray-vertex is in evolution

White- vertex is unvisited

Black- DFS has completed the processing

DFS can be finished with the help of Stack, LIFO execution. We start from node A, put into Stack, visited vertex is A. peek at the stack, vertex A has unvisited vertices B & C. Mark B as visited. At the top of the stack node B has D & E unvisited nodes. D marked as visited. Top of the stack D has no more unvisited nodes. Pop D from stack. Go to vertex E. visit E, unvisited vertices are E & I. Go to H visit its neighboring vertices, move on vertex I, visit it. I has unvisited vertices O & P. visit them. Continue this process when no vertices will be left unvisited.

Output generate will be in the form of,

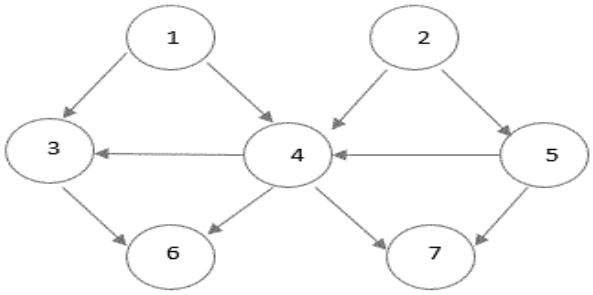


Fig. 16. Topological Sort.

A, B, D, E, H, L, M, N, I, O, P, C, F, G, J, K & Q

5.2.1 DFS Complexity

DFS complexity is O(V+E). For expressing the graph, if adjacency matrix is being used, than all edges adjacent to the vertex cannot find efficiently, the result will be in the form O (V^2) [8].

Another example:

Output: A B D E C F & G.

5.2.2 Ordering of Vertex

To generate linearly order of the vertices of the original graph, it's possible for us to use DFS. There are 3 collective methods we use to do this:

A pre-ordering generates order of vertices in the way that DFS algorithm first visited. To express the growth of the search, it's a natural & compressed technique. Preorder of the above example will be same of the output.

A post-ordering type of list of vertices in the way that they were last go through or visit by the algo. Post-order is D, E, B, F, G, C & A.

A reverse post-ordering generates list of vertices in an opposite order that they were visit last. Order is D, B, E, A, F, C & G. Parent node appear in the middle.

5.3 Topological Sort

In the early 1960's, topological sorting algorithms were 1st studied, (Jarnagin 1960) in the perspective of PERT approach for preparation in Project Management. In the

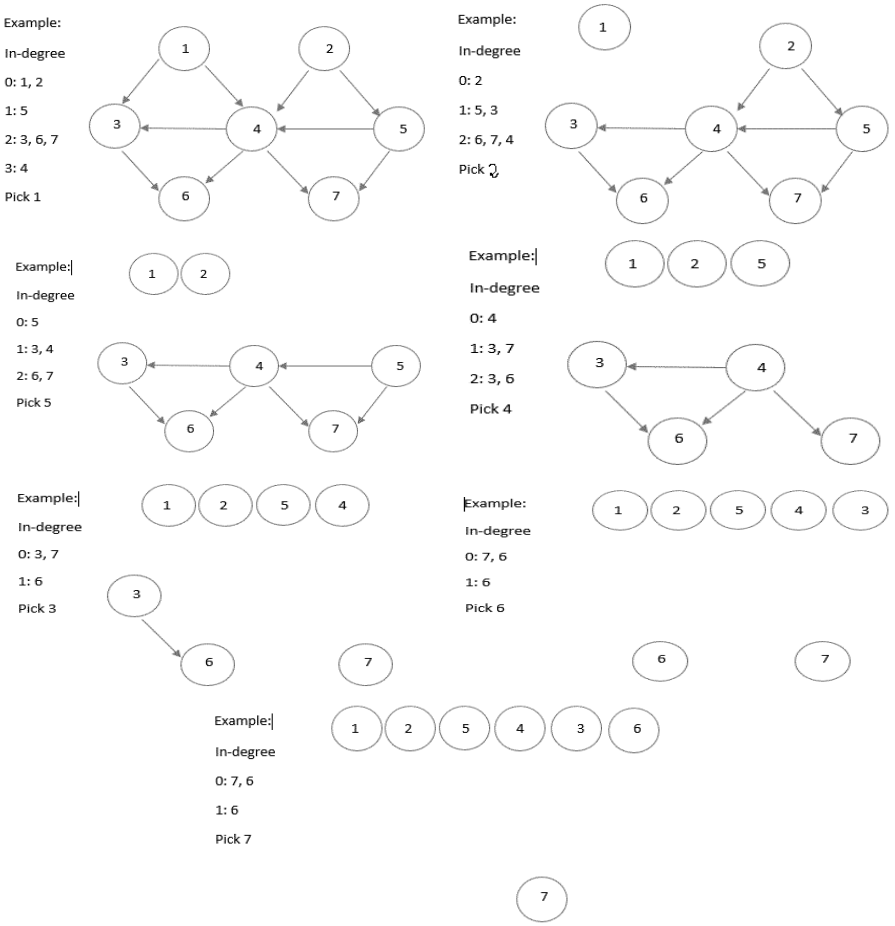


Fig. 17. Topo-Sort Example.

domain of computer science, sometimes topological sort of directed graph, called topological ordering or topo-sort, in which DG linear arrangement of the nodes is, parent must come before child, or we can say that from vertex u to vertex v, u must be come first before v. If graph contains no directed cycles, topological ordering will be possible only in this condition. To construct topological ordering of any DAG, algorithms are known will be in linear time for the development of topo-sorting. Through weighted directed acyclic graph to calculate shortest paths, topological ordering could be used. Using a serial algorithm, on a graph of m no of edges & n no of vertices, algo occupies linear time O (n+m) [8]. There are often many possible topological sorts of a DAG. For example

Topological order for this DAG will be in the form of:

– 1, 2, 5, 4, 3, 6, 7

– 2, 1, 5, 4, 7, 3, 6
– 2, 5, 1, 4, 7, 3, 6 etc.

Firstly make in-degree array, list all vertices with degrees. After that, vertices which value is in-degree 0 put into the queue. Now that vertex will be dequeue and insert into the output. In-degree array will be updated after visiting all nodes with 0 in-degree.

5.3.1 Topo-Sort Complexity

Pseudocode

```
1.  L - Initialize ordered list to be empty
2.  S - all vertices having no incoming edge's
3.  While (S ≠ ∅)
4.        Eliminate vertex v from set S
5.        add v to the end of the list L
6.  For each (vertex v1 with edge e from v to v1) do
7.        E, eliminate from graph
8.  If (v1 has no more incoming edges) then
9.         In set S add v1
10. If (graph contains edges) then, return error
11. Else return L ( sorted order)
```

In DAG, if all sets of nodes in an organized and sorted way joined by edges, then all these edges create a (DHP) directed Hamiltonian path, it's a topo-sort property. Topological sort order will be unique in the form if Hamiltonian path occurs. The usual algorithms for topo-sort having linear running time in the form of vertices + edges (links), in an asymptotic notation O (|V| + |E|).

5.4 Kahn's Algorithm

A type of algorithm, works by picking vertices in the same order just like the eventual topo-sort, 1st pronounced by Kahn (1962). Identify list of vertices without no incoming edges & insert them into the set. Solution contained in the list, if graph is DAG & if there exist at least one cycle, topo-sorting will be impossible.

5.5 Tarjan's Algorithm

A type of algorithm which we may use as an alternative for topological ordering based on DFS, 1st described by Tarjan (1976). Algorithm executes in linear time & every node & edge is visited once, in this algorithm.

6 Shortest Path Problem

To search a path, in the graph G, b/w two nodes or vertices, SPP is the problem, in which edge values sum will be minimized. SPP can be described for graphs whether

Pseudocode

```
1.  Dis [sor] ← 0
2.  For all ve ∈ V-{sor}
3.  For all ve
4.  do Dis [ve]←∞
5.  S←∅
6.  Qu←V
7.  while ( Qu ≠ ∅ )  do
8.  u ← mini-dis (Qu, Dis)
9.  S ← S u {u}
10. For all (v ∈ neighbors[u]) do
11. if Dis[ve] > Dis[u]+we(u,ve)
12. then Dis[ve]← Dis[u] + we(u,ve)
13. return Dis
```

weighted or unweighted. In weighted graphs, find a lowest cost path is basically the main objective. Un-weighted graphs, goal is to identify a path with smallest number of hops. The problem sometimes called single pair SPP, to differentiate it for the following variations: The single source SPP, for weighted graph find minimum weighted path from starting node to the end of the graph. Here we use some algorithms which based on the type of the graph, if graph is weighted we use Dijkstra algo otherwise use simple Breadth First Search for an unweighted graph. The single destination SPP, identify shortest pathways from every node to a specific ending node, in a weighted graph. In all pairs SPP, we need to find smallest routs b/w each group of nodes.

Early History of Shortest Path Algorithms

- 1955-Shimbel, Information Networks
- 1956-Ford, worked on economics of transportation (RAND)
- 1957-Johnson, Ladew, Seitz, Gray, Meaker, Leyzorek, Petry, Combat development department of Army Electronic Proving Ground,
- 1958-Bellman, Dynamic programming, Simplex technique for linear programming-Dantzig
- 1959-Dijkstra, fast & simple form of Ford's algorithm & Moore worked for Bell Labs

Applications of Shortest Path Algorithms

It's a beneficial model to solve problem. To find directions b/w physical locations, such as Google Maps, we use SP Algorithms. For this application fast specialized algorithms exist [12]. Other applications:

- Texture mapping.
- Routing of telecommunications messages.
- Subroutine in advanced algorithms.
- Optimal pipelining of VLSI chip [13].

– Urban traffic planning.

– Network routing protocols (OSPF, BGP, RIP).

– Robot navigation.

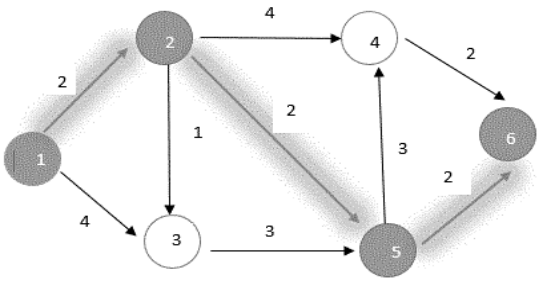


Fig. 18. Dijkstra Algorithm

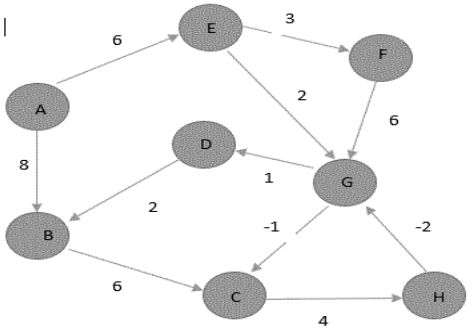


Fig. 19. Directed Graph for Bellman-Ford.

– Approximating piecewise linear functions.

6.1 Dijkstra Algorithm

In a graph, to search route from starting node to the destination node we use DA, with undirected edges & non-negative edge values. Connected graph is compulsory. It uses a greedy approach to find shortest path weights in the graph.

In line 1, distance to source vertex is 0. In line 3, declare all distances to ∞. Line 4, S, visited vertices set is initially {}. Line 5, queue initially comprises of all nodes. Line 6, while queue is ≠ {}. Line 7, choose variable from queue having min-distance. Line 8, in the list of visited nodes add u. Line 10, if new shortest track found, line 11- update value of shortest path.

Pseudocode

```
1. ds=0
2. dv = ∞ ∀ (v ≠ s)
3. j=1;
4. For  j to m-1
5. ∀ edge u → v of cost c
6. dv = minimum (dv, du + c)
   ∀ edge u → v of cost c
7. if (dv > du + c)
   return false
8. return true
```

In the following graph, if we select path {1, 2, 4 & 6} then its total cost will be 8. Now, go from another path {1, 3, 5 & 6} its total cost is equal to 9. If we go from {1, 2, 3, 5 & 6} its cost is equal to 8. Another path {1, 2, 3, 5, 4 & 6} cost is equal to 11. Another possibility {1, 2, 5, 4 & 6} cost will be 9. If we go from {1, 2, 5 & 6} its cost will be 6. This is the lowest cost from all costs, so Dijkstra algorithm choses way with lowest cost values and uses greedy approach to reach at the destination vertex.

6.2 Bellman-Ford Algorithm

In a weighted digraph, in which we start searching to identify lowest value route from a single start node to all of the other vertices of the graph [19]. Dijkstra algorithm is faster from BFA. But BFA is beneficial to control graphs with negative weights. There can be situations, where a graph can be in negative weight cycles, but we do not see these situations in a real life. But now there are some complex situations with negative edges. Few of them are detecting network failures or linear programming.

6.2.1 Bellman Ford Algorithm Description

Maintain list of unvisited nodes

Select source node & allocate maximum cost infinity to every other node

Cost of the source node remains 0 as it actually takes nothing to reach from the start node to itself

In every iteration of the algo it tries to minimize the cost of vertex

Repeat step 4 for |v|-1 times. From the last iteration we will have a shortest path from start to every vertex.

1st step

Suppose A is a starting node with cost 0, insert all vertices (A, E, F, D, G, B, C & H) in the list. Allocate cost infinity all vertices except starting vertex A.

2nd Step

– (A, E): cost of E [6].

– (A, B): cost of B [8].

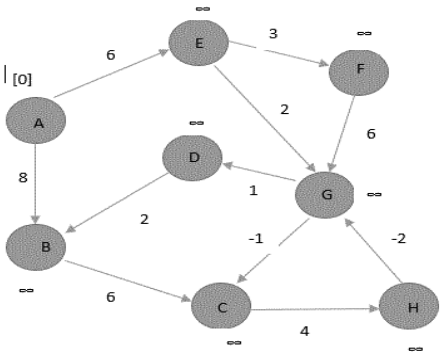


Fig. 20. 1st Step BFA.

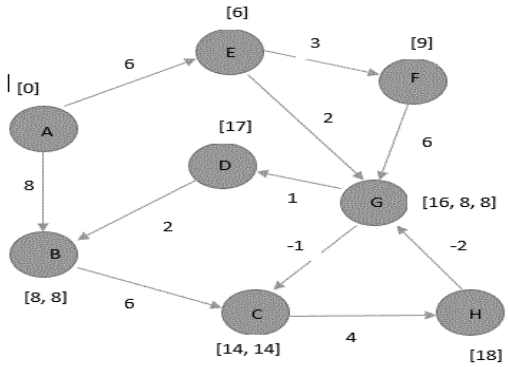


Fig. 21. 2nd Step BFA.

– (B, C): cost of C [14].
– (C, H): cost of H [18].
– (H, G): cost of G [16].
– (G, C): cost of C [14].
– (G, D): cost of D [17].
– (D, B): cost of B [8].
– (E, F): cost of F [9].
– (E, G): cost of G [8].
– (F, G): cost of G [8].

3rd Step

– (A, E): cost of E [6].

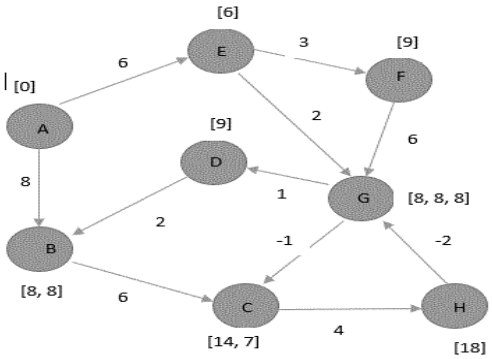


Fig. 22. 3rd Step BFA.

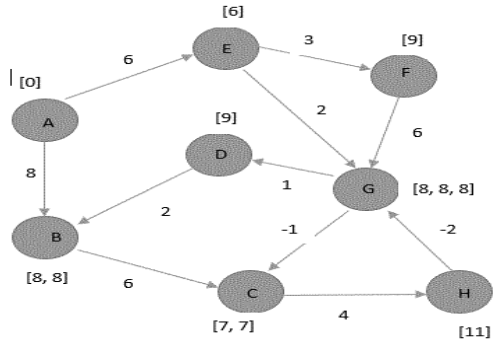


Fig. 23. 4th Step BFA.

- (A, B): cost of B [8].
- (B, C): cost of C [14].
- (C, H): cost of H [18].
- (H, G): cost of G [8].
- (G, C): cost of C [7].
- (G, D): cost of D [9].
- (D, B): cost of B [8].
- (E, F): cost of F [9].
- (E, G): cost of G [8].
- (F, G): cost of G [8].

4th Step

- (A, E): cost of E [6].

Pseudocode

```
1. for (m=1 to v, m++) do
2. for (n=1 to v, n++) do
3. Dis(m, n)= wei(m, n)
4. for (o=1 to v) { //o is an intermediate vertex
5. for (m=1 to v) do {
6. for (n=1 to v) do {
7. If (dist(m,o) + dist (o,n) < dist pass n,m )
8. Then dist pass n,m = dist pass m,o + dist pass o,n }}}
```

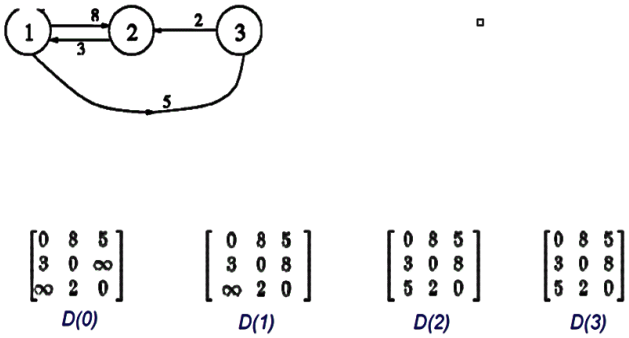


Fig. 24. Floyed Warshall Algorithm.

- (A, B): cost of B [8].
- (B, C): cost of C [7].
- (C, H): cost of H [11].
- (H, G): cost of G [8].
- (G, C): cost of C [7].
- (G, D): cost of D [9].
- (D, B): cost of B [8].
- (E, F): cost of F [9].
- (E, G): cost of G [8].
- (F, G): cost of G [8].

Complexity of Bellman ford could be represented in the form of O(VE) time, declaration will take O(V) time in 1st line, in the line 2 to 4 take O(E) time, V-1 passes over the edges & in line 5 to 7 for For loop takes O(E) time.

6.3 Floyd Warshall Algorithm

FWA is an algorithm in which, graph has –ve or +ve edge values but not containing –ve cycles. It finds only the length not the path. FWA is an example of dynamic

Pseudocode

```
JSS (V, E, s)
1.  begin
2.  Q=V
3.  for all (ver ∈ Q) do
4.          l[ver] = ∞
5.                  l[s]=0
6.  Do- while (Q ≠ ∅)
7.  begin:
8.       h= extr-min(Q)
9.  For ver ∈ adja [h] -- do
10. If codition  ver ∈ Q l[h] + we(h,ver) < l[ver]
11. Then l[ver] = l[h] + we(h,ver)  end while -- end of JSS
```

programming. This algorithm also known as Roy-Warshall Algorithm, Floyd's Algorithm, Roy-Floyd Algorithm & WFI Algorithm. FWA running time is O (V^3).

If (j=k), G[j][k] is 0,

If no edge b/w j to k then, G[j][k] is ∞

6.4 Johnson's Algorithm

JA is an algo to search shortest tracks b/w all pair of vertices in an edge-weighted, directed or sparse (a graph with few edges) graph. It permits some edge weights to be negative values, but no –ve cycles. It works by using bellman-ford & Dijkstra algorithm. Time complexity of this algo is O (V^2logV + VE).

7 Graph Coloring

Graph coloring has been using in many real time applications of computer science & it is the most significant part of graph theoretical concepts. Numerous labelling techniques exists & can be applied on the necessity bases. The appropriate graph coloring is the overall coloring of links & nodes (V & E), in a way no two nodes consists of the matching color, and must be used smallest no of colors. The usage of smallest no of colors in the graph called (CN) chromatic number & graph G called (PCG) properly colored graph. From 1972, Chromatic numbering problem we said one of the NP-complete problem. Register allocation in compilers is the major application of graph coloring and was introduced in 1982. Graph coloring has been using in theoretical concepts as well as practical applications, but there are different limitations that can be set on graphs, they may be color assigning or can be color itself. Due to limitations, it is still a vigorous domain of research [18].

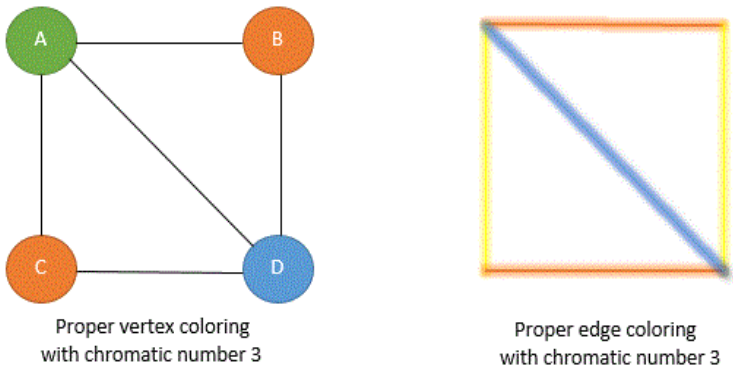


Fig. 25. Graph Coloring.

7.1 Vertex Coloring

Vertex coloring is the most famous problem in graph coloring. The problem is, color each vertex in a way that two adjacent vertices does not consist of the matching color.

The other graph filling complications like Link (Edge) and Map labelling could be converted into node coloring. Graph coloring G is basically a mapping (plotting) function & denoted as c: V (G)--S.

In set S, Variables of S are colors, assign color to the nodes, pick colors from a color class S. C is K-coloring, if |S|=K. if adjacent nodes have diverse colors then graph coloring will be appropriate . G called K-colorable, if G has appropriate K-coloring. For the representation of chromatic number, notation X(G) is used. G will be K-chromatic, If X (G) = K [14].

7.2 Edge Coloring

An EC is an appropriate filling of the edges, which means no two edges consists of the same color, when we allocate colors to edges. Edge CN or CI (chromatic index) X' (G), is lowest no of colors needed for EC. For (3-EC) of a cubic graph Tait coloring is basically used.

7.3 Total Coloring

No neighbor edges, no neighbor vertices, no edges & vertices allocated the same color, in this way total coloring will always be supposed to be proper. A kind of coloring applied on nodes of graph & links. X^n (G) notation represents the Total Chromatic Number (TCN) of a graph.

7.4 Chromatic Number

Pseudocode

```
1: function Main
2: D = ∅
3: for all v ∈ V do
4: D ∪ v. out degree
5: end for
6: while (D ≠ ∅) do
7: d ← max {d | d ∈ D}
8: schedule(vertices of degree(d))
9: D − {d}
10: end while
11: execute scheduled()
12: end function
14: return ∀v ∈ V : v → v. color
```

Finding a chromatic number is an NP-complete problem. Since we do not have any idea, how many colors will be used in the graph? For graph coloring, minimum no of colors represent Chromatic number.

8 Graph Coloring Algorithms

The problem of sequentially coloring an arbitrary graph has been studied widely. 4-coloring problem exist for planner graphs, but non-planner graphs may require large no of colors. To color a graph by using minimum number of colors, looks a very simple problem, but in reality it's not a simple one. To solve this problem, no of serially based polynomial time algorithms exists, which basically use limited no of colors. Here we present some sequential GC algorithms like LDF algorithm and smallest degree last algorithm and SDO and IDO algorithms but SDO and IDO are not appropriate to parallelization. [16]

For graph labeling, there exists a lot of heuristic techniques & one is Greedy Graph Coloring [17]. This type of approach emphasis on choosing the next vertex for coloring. In this heuristic based technique, once a node is filled with a color, then color cannot be changed. Here we present greedy algorithm for graph coloring, we cannot sure it will use minimum number of colors or not. In the graph, extreme degree of a node is d, so basic greedy coloring algorithm cannot uses more than d+1 colors.

– Assign color to the 1st vertex in the graph.
– Do step 3 when V-1 nodes remained.
– Suppose, when we choose a node and assign color, make sure that color has not been used previously labeled nodes, adjacent to that vertex. If any condition exist, then allocate new color.

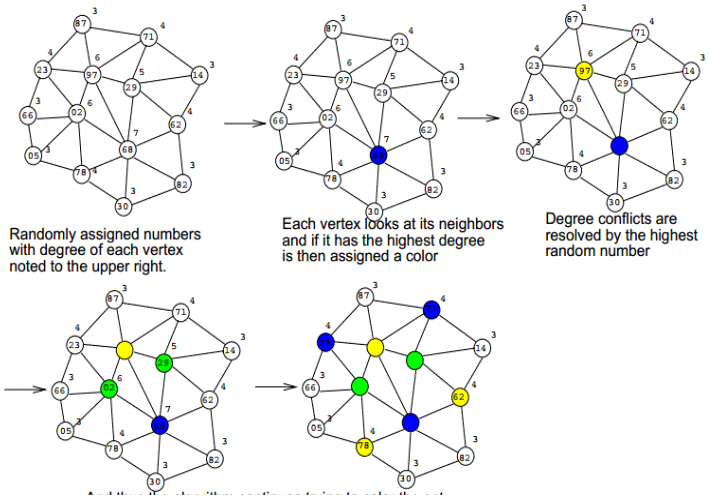


Fig. 26. Largest Degree First.

Pseudocode

1. L = 1
2. k = 1
3. U = V
4. while ([U] > 0) do
5. while { vertices v ∈ U with dU (v) <= L } do in parallel
6. S = { all vertices v with dU (v) <= L }
7. for all vertices v ∈ S , w (v) = k
8. U = U S
9. k = k + 1 end do
10. L = L + 1 end do

8.1 First Fit (FF)

FF algorithm is the fastest & easiest approach. In this algorithm, algorithm allocates each vertex smallest legal color in sequence. Complexity of this algo is expressed in linear time O (n) [14].

8.2 Degree Based Ordering (DBO)

DBO offers an enhanced tactic for graph coloring. For selecting the node to be labelled, it uses a specific selection norm. It is better than first fit because in which from random direction, we merely pick a node. For choosing a next node to be colored, few techniques have been suggested like:

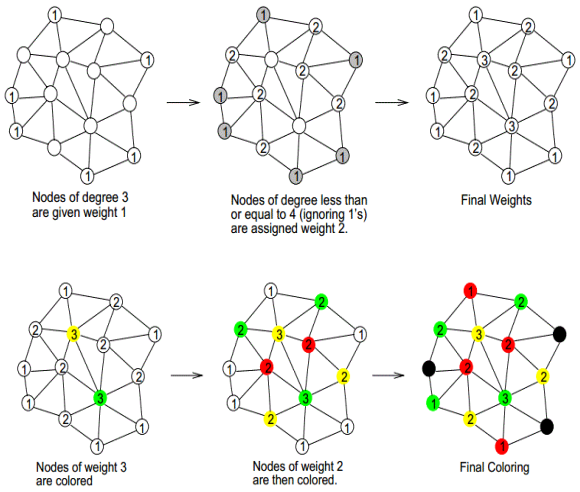


Fig. 27. Smallest Degree Last.

8.2.1 Largest Degree Ordering (LDO)

In this algorithm, vertices will never be colored in random order but vertices with the largest degree being the colored first. This heuristic offers better coloring then first fit and can be implemented in the notion O (n^2) [14].

In this example, we color those vertices having largest degree, we will never randomly choose the node. After selecting node, color it [15].

8.2.2 Smallest Degree Last Algorithm

This algorithm works in two steps, a weighting step and coloring step. The weighting step starts by searching small degree node. After assigning the weights, in coloring step, color the largest value node in the graph.

In the above example, in the 1st graph we will find the node having minimum no of edges. So nodes of degree 3 assign weight 1. In the 2nd figure, nodes of degree less than or equal to 4 neglect 1 assigned weight 2. Now we will search the highest value weight, color them [14-15].

8.2.3 Saturation Degree Ordering (SDO)

The SD of a node demonstrated by means of the no of its neighbors inversely shaded nodes. SDO heuristic offers superior shading then Largest Degree Ordering and could be denoted in the notion of O (n^3) [15].

8.2.4 Incident Degree Ordering (IDO)

IDO is the reformed form of SDO heuristic. Incident degree of a node demonstrated as like the no of its neighbor's painted nodes. Heuristic denoted in the notion of O (n^2) [15].

Pseudocode

1. while (no of shaded vertices < m) {
2. maximum = -1, I= 1
3. loop I to m {
4. if condition !colored(n_i) {
5. d equal to Degree(n_i)
6. if condition d greater than maximum{
7. maximum equal to d
8. ind equal to i}
9. if condition d equal to maximum
10. if condition ID(n_i) greater than ID(n_{index})
11. ind equal to i}
12. Color (n_{index})
13. no of colored nodes= no of colored nodes + 1 } }

Pseudocode

1. while (no of colored nodes < m) {
2. Maximum equal to -1, I equal to 1;
3. Loop I to m {
4. If condition !colored(n_i) {
5. d equal to SD(n_i)
6. if condition d greater than maximum {
7. maximum equal to d
8. ind equal to I }
9. if condition d equal to maximum
10. if condition Degree(n_i) greater than Degree(n_{index})
11. ind equal to I }
12. Color (n_{index})
13. No of colored nodes = no of colored nodes + 1}} [14]

8.2.5 New Heuristic Coloring Algorithms

In the first algorithm, (modification of LDO), we improved the LDO algorithm by merging it with an IDO algorithm. The algorithm operates like LDO, IDO was used to select nodes having same number of degree. For picking next nodes to be colored, there are two tactics.

– Number of nodes linked to the node Largest Degree Ordering.

– Number of colored nodes joined to the node Incident Degree Ordering.

In the second algorithm, (modification of SDO), we merge the both SDO algorithm and LDO. The algorithm functions like as SDO, when we initiate if there are 2 nodes having

same degree, then LDO we used to select nodes b/w them with the same degree. We applied 2 tactics for picking next nodes to be filled with colors.

– No of colors close to the node SDO.
– No of vertices nearby the node LDO.

8.2.6 Applications of Graph Coloring

The following applications of graph coloring can be mentioned.

– Making schedule: Problem is graph coloring in the form, assume, we have a list of students and subjects and we want to make an exam schedule for university. Many subjects would have common students. How we will manage that no two exams of the same student subject are scheduled at the same time? This problem we can model with the help of graph in which each subject will be the vertex and an edge b/w them we say there exists a common student.
– Bipartite Graphs: With the help of using colors we can model graph is bipartite or not. Graph will be bipartite if it is two-colorable [18].
– Sudoku: It is also a variation of graph coloring problem where each cell is denoted by a vertex. There is an edge b/w 2 vertices if they are in same column or same block or same row.
– Register allocation: We have already said, register allocation in compilers is a graph coloring problem. Problem is, it is a method of allocating huge number of objective program elements onto a minor number of Central Processing Unit records (registers).
– Map coloring: Geographical map of cities, states and countries where 2 adjacent cities cannot be represented with the same colors.

9 Conclusion

Algorithms are the foundation of modern sciences and empirical theories. Researches made so far significantly improve and insist upon further areas to be examined and dissected within algorithms, graph theory and their nexus.

The applications of algorithms go way beyond what we have though and researched about. Graph theory also plays a crucial part in expressing, authenticating and implementing these sciences and theories.

After analyzing the formalization of both disciplines in details and consulting researches performed on both the domains, it has been concluded that a combination of these two can open new gates to the world of sciences in terms of modelling and computational devices, communications, data organization and searching techniques.

References

1. Abdul, M., Ibtisam, R.: Graph Theory: A Comprehensive Survey about Graph Theory Applications in Computer Science and Social Networks. Inventions, 5(10) (2020). doi: 10.3390/inventions5010010.
2. Rodrigue, J.P.: The Geography of Transport Systems. Routledge (2020)
3. Gross, J.L., Yellen, J., Anderson, M.: Graph Theory and its Applications. Chapman and Hall/CRC (2018)
4. Deo, N.: Graph Theory with Applications to Engineering and Computer Science. Courier Dover Publications (2017)
5. Fernández, F.M., Castro, E.A.: Algebraic Methods in Quantum Chemistry and Physics. CRC Press (2020)
6. Mondal, B., De, K.: An Overview Applications of Graph Theory in Real Field. International Journal of Scientific Research in Computer Science, Engineering and Information Technology, 2(5), pp. 751–759 (2017)
7. Tran, N.: Review of the Algorithm Design Manual. ACM SIGACT News, 53(3), pp. 21–23 (2022). doi: 10.1145/3561064.3561068.
8. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms. MIT Press (2022)
9. Bhargava, V.R., Assadi, P.: Hiring, Algorithms, and Choice: Why Interviews Still Matter. Business Ethics Quarterly, 34(2), pp. 201–230 (2024). doi: 10.1017/beq.2022.41.
10. Even, S.: Graph Algorithms. Cambridge University Press (2011). doi: 10.1017/CBO9781139015165.
11. Goodrich, M.T., Tamassia, R., Mount, D.M.: Data Structures and Algorithms in C++. John Wiley & Sons (2011)
12. Sanders, P., Schultes, D.: Engineering Fast Route Planning Algorithms. In: International Workshop on Experimental and Efficient Algorithms, pp. 23–36 (2007). doi: 10.1007/978-3-540-72845-0_2.
13. Chen, D.Z.: Developing Algorithms and Software for Geometric Path Planning Problems. ACM Computing Surveys (CSUR), 28(4), pp. 18 (1996)
14. Baghel, M., Agrawal, S., Silakari, S.: Recent Trends and Developments in Graph Coloring. In: Proceedings of the International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA), pp. 431–439 (2013). doi: 10.1007/978-3-642-35314-7_49
15. Allwright, J.R., Bordawekar, R., Coddington, P.D., Dincer, K., Martin, C.L.: A Comparison of Parallel Graph Coloring Algorithms. SCCS-666, pp. 1–19 (1995)
16. Husfeldt, T.: Graph Colouring Algorithms. arXiv Preprint (2015). doi: 10.1017/CBO9781139519793.016.
17. Shukla, A.N., Bharti, V., Garg, M.L.: A Greedy Technique Based Improved Approach to Solve Graph Colouring Problem. EAI Endorsed Transactions on Scalable Information Systems, 8(31) (2021). doi: 10.4108/eai.16-2-2021.168716.
18. Solomon, S., Wein, N.: Improved Dynamic Graph Coloring. ACM Transactions on Algorithms (TALG), 16(3), pp. 1–24 (2020). doi: 10.1145/3392724.