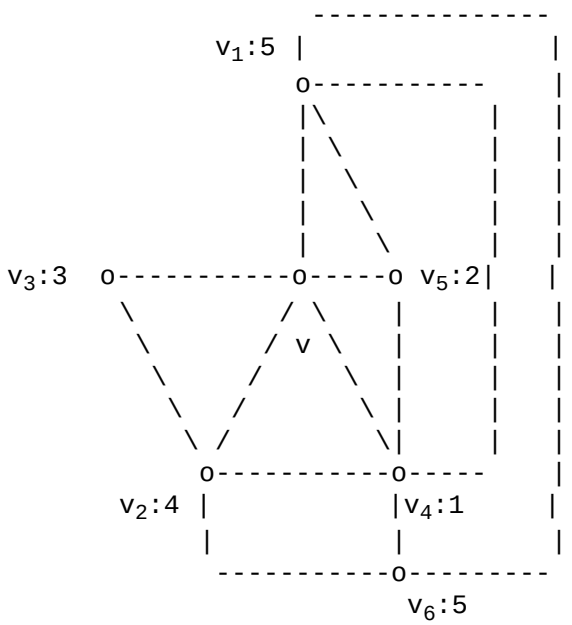


Figure 7-9. G - v with Recursively Defined Five Coloring.

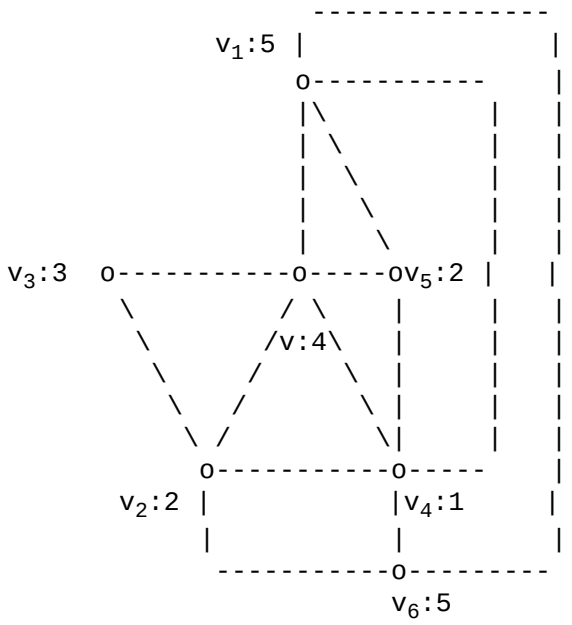


Figure 7-10. Coloring of G after Modification of G - v.

9	1	1	2	4(0)	0	BACKUP	3
10	1	1	3	0	0	ADVANCE	4
11	1	1	3	2	0	ADVANCE	5
12	1	1	3	2	4(0)	BACKUP	4
13	1	1	3	3	0	ADVANCE	5
14	1	1	3	3	2	DISPLAY	5
15	1	1	3	3	4(0)	BACKUP	4
16	1	1	3	4(0)	0	BACKUP	3
17	1	1	4(0)	0	0	BACKUP	2
18	1	2	0	0	0	ADVANCE	3

v_i column: gives value of VCR(i) at end of stage.

k column: gives value of parameter k at end of stage.

4(0): indicates Next returns 4, which backtrack conrol then resets to 0, as required for BACKUP action.

Figure 7-8. Partial Trace of Backtrack Vector for for Graph of Figure 7-5 where M = 3.

Figure 7-6. Backtrack Tree with M = 2.("-" indicates deadend.)

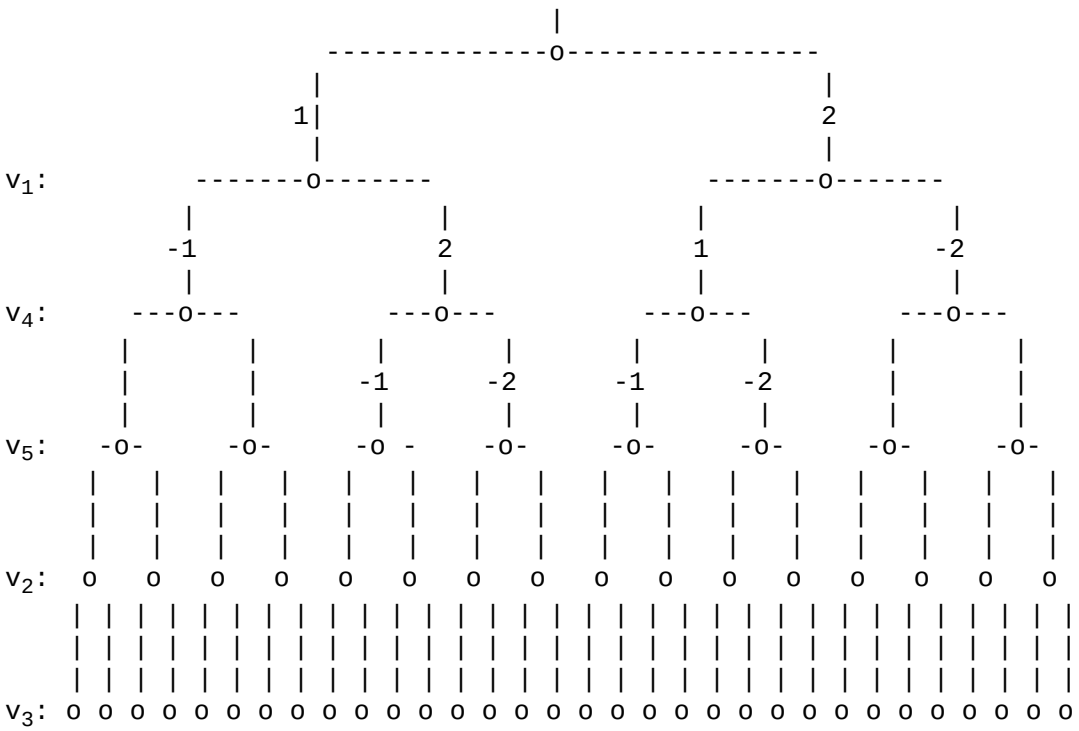


Figure 7-7. Search Tree for Different Scan Order.

Stage	v_1	v_2	v_3	v_4	v_5	ACTION	k
0	0	0	0	0	0	ADVANCE	1
1	1	0	0	0	0	ADVANCE	2
2	1	1	0	0	0	ADVANCE	3
3	1	1	2	0	0	ADVANCE	4
4	1	1	2	2	0	ADVANCE	5
5	1	1	2	2	3	DISPLAY	5
6	1	1	2	2	4(0)	BACKUP	4
7	1	1	2	3	0	ADVANCE	5
8	1	1	2	3	4(0)	BACKUP	4

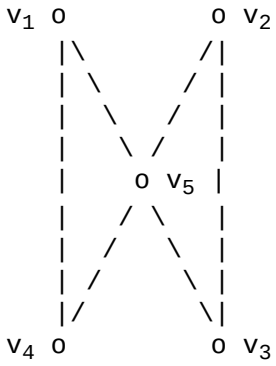
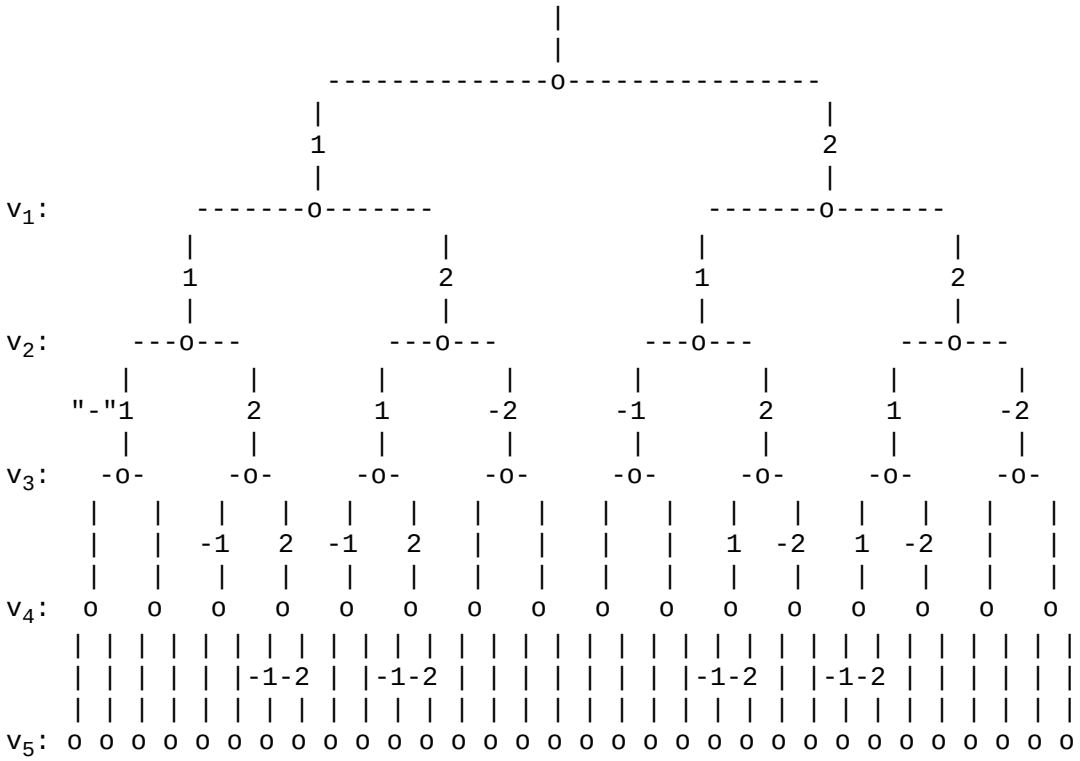


Figure 7-5. Graph for Backtracking Example.



First: T_1, T_4
Second: T_2
Third: T_3

(c) Shortest Length Schedule.

Figure 7-3. Scheduling Example.

SECTION 7-3 BACKTRACK ALGORITHM

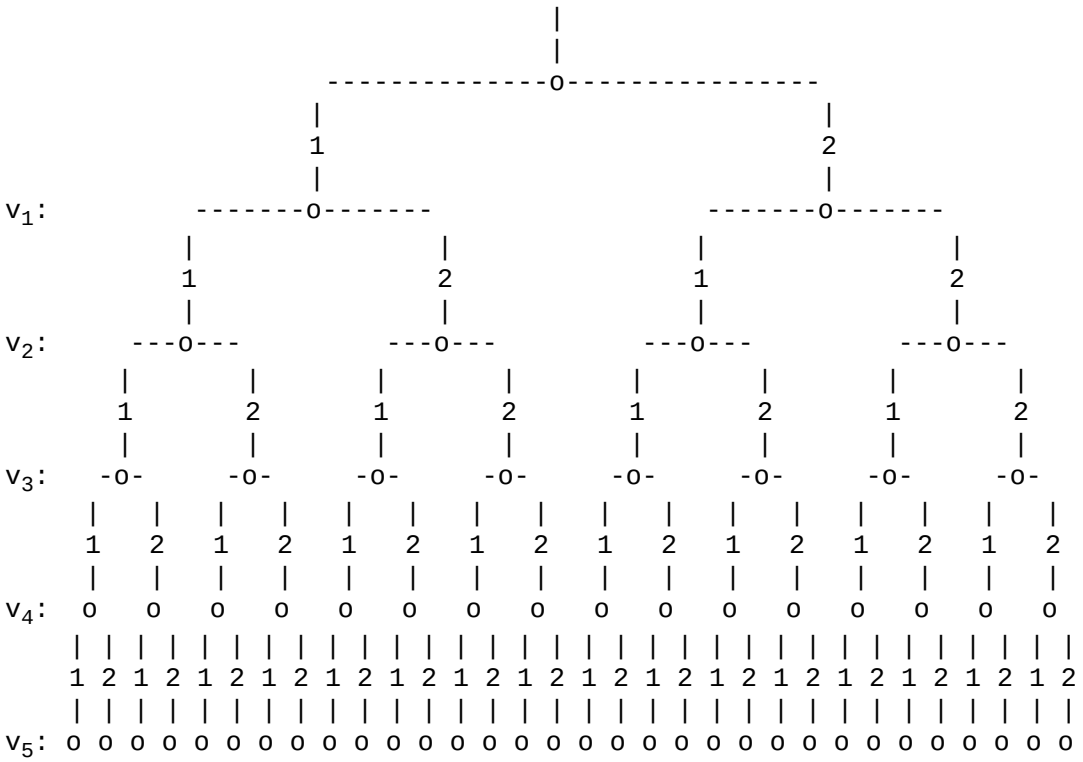


Figure 7-4. Tree of Colorings: |V| = 5, M = 2.

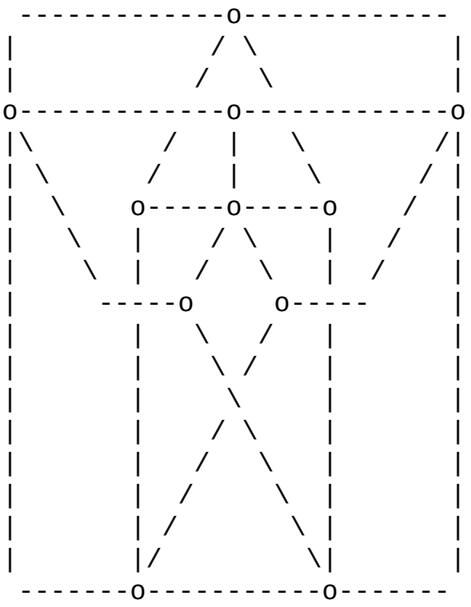


Figure 7-2. Smallest Triangle-Free Four-Chromatic Graph.

SECTION 7-2 MODELS: CONSTRAINED SCHEDULING AND ZERO-KNOWLEDGE PASSWORDS

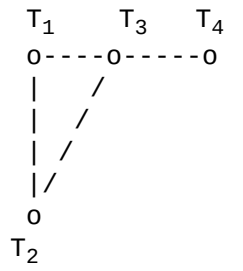
T_1: T_2, T_3

T_2: T_1, T_3

T_3: T_1, T_2, T_4

T_4: T_3

(a) Task Conflict List.



(b) Conflict Graph.

are scanned. Can you suggest a rationale for a preferred order for scanning?

(12) Color each of the edges of K(6) either red or blue. Then, prove the resulting graph contains a monochromatic triangle, that is, a cycle of length 3 all of whose edges have the same color.

(13) Implement the recursive version of the five-color algorithm.

(14) The following algorithm approximately colors a three-chromatic graph G(V,E).

(a) **Set** n to |V| and color to 1.
(b) **while** max(G) $\geq n^{(1/2)}$
do Select a vertex v of maximum degree
Color the Adj(v) with color and color + 1
Increment color by 2
Set color(v) to color
Set G to G - v - Adj(v).
(c) Use the greedy procedure given for the weakened form of Brooks' Theorem to color any remaining vertices.

Give the performance of this algorithm and prove it colors a three-chromatic graph with at most $3n^{(1/2)}$ colors. What happens if the algorithm is applied to a k-chromatic graph? (See Wigderson (1983) for a more general discussion.)

CHAPTER 7: GRAPH COLORING

SECTION 7-1 BASIC CONCEPTS

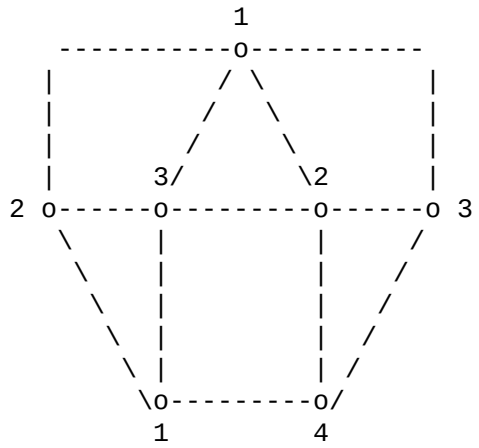


Figure 7-1. A Four-Chromatic Graph.

CHAPTER 7: REFERENCES AND FURTHER READING

See Behzad, et al. (1979) for a detailed discussion of coloring problems and an overview of the proof of the Four-Color Theorem. Parenthetically, the proofs of theorems in Behzad, et al. are exceptionally clear, and several have served as the basis for our own proofs. See also Appel and Haken (1977) for a popular review of their groundbreaking proof. A faster (linear) time version of the five-color algorithm is given in Chiba, Nishizeki, and Saito (1981). The approximate coloring algorithm described in the exercises, and a generalization, is given in Wigderson (1983). See Capobianco and Molluzzo (1978) for interesting examples, such as on the unique coloring problem.

CHAPTER 7: EXERCISES

(1) Determine the vertex and edge chromatic numbers of each of the five regular polyhedra.

(2) Let G(V,E) be a connected cubic graph of order > 4, and girth = 3. Determine the VCHR(G).

(3) Let G be bipartite. Prove that the vertex chromatic number of the complement of G equals the order of the maximum order complete subgraph in G^c.

(4) A graph G(V,E) is said to be *uniquely n-colorable* if VCHR(G) equals n, and every n-coloring of G induces the same partition of the vertices of G. Prove that if G is uniquely n-colorable, G is (n - 1)-connected.

(5) Prove that if G is bipartite, ECHR(G) = max(G).

(6) A graph G(V,E) embedded in the plane is said to be *n-region-colorable* if its regions can be colored with at most n colors, so that adjacent regions are colored differently. Prove that every planar graph is four-colorable if and only if every planar embedding of a graph is four-region-colorable.

(7) Prove that if G is connected, not an odd cycle, and all its cycles have the same parity, ECHR(G) = max(G).

(8) Prove that if G is a nontrivial regular graph of odd order, ECHR(G) = max(G) + 1.

(9) Write a program that simulates the operation of a zero-knowledge password system using the graph passwords method described in the text.

(10) Write a program that performs constrained scheduling and uses a backtracking algorithm to find an optimal schedule.

(11) Use Monte Carlo estimation (see Chapter 2) to estimate the performance of a backtracking algorithm for finding the k-colorings of a graph. Compare the number of vertices in the estimated backtrack tree for the colorings with the number of vertices in the actual backtrack tree for the colorings. Examine the dependence of the performance on the order in which the vertices

The c|V| term comes from the time it takes to locate and delete a vertex v of degree at most 5, and the time to find a pair of vertices i and j disconnected in the induced subgraph. Iteration gives $O(|V|^2)$ performance.

ny recurive algorithm can be unravelled into an iterative algorithm, which may be more efficient but more cluttered than the recurive version. To convert the five-color algorithm to an iterative form, we first explicitly identify the vertices R(k) of degree at most five used at each stage of the recursive algorithm. We then color a small initial graph directly and extend the coloring by successively adding back the previously removed vertices R(k), recoloring the intermediate graphs as we proceed, just as in the recursive algorithm. The iterative version follows.

```
Procedure Five-Color (G)

(* Iterative Five-color algorithm *)

VAR   G, G_copy: Graph
      R(|V(G)| - 5), i, j, w:  Vertex identifier
      C_1, C_2: Component identifier
      k: 1..|V(G)| - 5

Set G_copy to G

for k = 1..|V(G_copy)| - 5  do  Find R(k) in G of degree ≤ 5
                                Set G to G - R(k)

Assign initial coloring to G

for k = |V(G_copy)| - 5  to  1  do

   if    neighbors of R(k) use 5 colors

   then  Find neighbors i and j of R(k) from distinct components
         C_1  and C_2 of the subgraph induced by the vertices
         colored the same as i and j

         for each w in C_1 do if    Color(w)     =  Color(i)
                              then  Set Color(w) to Color(j)
                              else  Set Color(w) to Color(i)

   Set Color(R(k)) to color not used by neighbors of R(k)

   Add R(k) back to G using G_copy

End_Procedure_Five_Color
```

```
Procedure Five_Color (G)

(* Returns five-coloring of planar G *)

var   G: Header
      i, j, v, w:  Vertex pointer
      C1, C2: Component identifier
      Nextcount: Integer function
      Next: Boolean function

if    |V(G)| ≤ 5

then  while Next(G,w) do Set Color(G,w) to Nextcount

else  Select v in V(G) of degree at most 5

      Five-Color(G - v)

      if   Adj(v) uses 5 colors

      then (* Modify coloring of G - v *)

         Find i and j in Adj(v) from distinct components C1 and
         C2 of the subgraph induced by the vertices colored the
         same as i and j

         for each w in C1 do if    Color(w)     =  Color(i)
                             then  Set Color(w) to Color(j)
                             else  Set Color(w) to Color(i)

      Set Color(v) to color not used in Adj(v)

End_Procedure_Five_Color
```

An example is shown in Figures 7-9 and 7-10.

Figures 7-9 and 7-10 here

We used a planar representation of the graph in the proof when we assumed we knew a clockwise ordering of the vertices neighboring v, but we do not need a planar representation in the algorithm. We only need to find a pair of vertices neighboring v which are in different components of their color induced subgraph. That is, we test each pair of neighbors i and j to determine whether or not they are connected in the induced subgraph on Color(i) and Color(j). We have to test at most ten pairs since v has only five neighbors. We can perform each test in O(|E|) time using depth first search. Since the graph is planar, |E| = O(|V|), and so each test is actually O(|V|); so all ten tests take only O(|V|) time.

If we denote the time required by Five_Color(G) by f(|V(G)|), then f satisfies the recurrenc:

$$f(|V(G)|) = f(|V(G)| - 1) + c\ |V|.$$

If the neighbors of v use all five colors, we first modify the coloring of G - v so the neighbors use only four colors after which we can again extend it to a five-coloring of G. Let us define H(i,j), i,j = 1, ..., 5, as the subgraph induced in G - v by the vertices of G - v colored i and j. For simplicity, assume the neighbors of v are arranged in clockwise order around v in some planar representation of G are v_1, ..., v_5, and that v_i has color i. There are two cases according to whether or not v_1 and v_3 lie in different components of H(1,3).

v_1 and v_3 in different components of H(1,3)

Let v_i (i = 1, 3) lie in component $H_i(1,3)$ of H(1,3). If we invert the colors in $H_2(1,3)$, changing every color-1 to color-3, and color-3 to color-1 and leave the colors in $H_1(1,3)$ unchanged, we obtain a valid coloring in which v_1 and v_3 are both color-1; so we are free to color v with 3.

v_1 and v_3 in same component of H(1,3)

There must be a path from v_1 to v_3 in H(1,3). Together with the edges (v,v_1) and (v,v_3), this path determines a cycle C in G. Because of the clockwise arrangement of the neighbors of v, one of the vertices v_2 and v_4 must lie in the interior of this cycle and one must lie in its exterior. Consequently, v_2 and v_4 cannot lie in the same component of H(2,4), since any path in H(2,4) from v_2 to v_4 would have to pass through C - v, which is impossible since the vertices of C - v have colors 1 or 3. Therefore, v_2 and v_4 fall under the previous case. This completes the extension of the coloring of G-v to a coloring of G, and so completes the proof of the Five-Color Theorem.

FIVE-COLOR ALGORITHM

The inductive proof leads naturally to a recursive algorithm. We can represent the graph as a linear list with vertices.

```
type  Vertex = record
                 Index: 1..|V|
                 Color: 0..5
                 Degree: 0..|V| - 1
                 Successor: Vertex pointer
                 Edge List: Edge pointer
               end
```

In addition to its usual fields, each vertex has a color, which is initially 0, and a degree. The other types are defined as usual. Next(G,w) returns in w a pointer to the record for the next vertex in G, and fails if there is none. A high level description of the procedure follows.

```
Procedure Color_Backtrack (G(V,E), M)

(* Lists all colorings of G using ≤ M colors *)

var  G: Graph
     M: Integer
     k: 1..|V|
     VCR(|V|): 0..M + 1
     Exit: Boolean

Set  VCR (1..|V|) to 0
Set  k to 1
Set  Exit to False

repeat

   Next(VCR,k)
   case
         (* CONDITION *)                 (* ACTION  *)

  1. VCR(k) ≤ M and  k < |V|:  Set k to k + 1      (*ADVANCE*)

  2. VCR(k) ≤ M and  k = |V|:  Display VCR(1..|V|) (*DISPLAY*)

  3. VCR(k) > M and  k > 1:    Set VCR(k)   to 0   (*BACKUP*)
                               Set k to k- 1

  4. VCR(k) > M and  k = 1:    Set Exit  to True   (* EXIT *)

until Exit
End_Procedure_Color_Backtrack
```

7-4 FIVE COLOR ALGORITHM

Planar graphs are four-colorable, but the proof of the Four-Color Theorem is profoundly difficult. (The current proof has over a 1,000 special cases which required over a 1,000 hours of computer time to analyze.) The proof of the Five-Color Theorem is rather simple and serves as the basis of an $O(|V|^2)$ five-coloring algorithm. A more elaborate O(|V|) five-color algorithm for planar graphs is described in Chiba, Nishizeki, and Saito (1981).

PROOF OF FIVE-COLOR THEOREM

The proof is by induction on |V(G)| and relies on the existence of a vertex v in G of degree at most 5. We will assume that G - v is five-colorable and will show how to modify and extend its coloring to a five-coloring of G. We denote the colors by 1..5.

If the neighbors of v in G are colored using only the four colors in G - v, we extend the coloring from G - v to G by coloring v with a fifth color not used by any of its neighbors.

Stage	v_1	v_2	v_3	v_4	v_5	ACTION	k
0	0	0	0	0	0	ADVANCE	1
1	1	0	0	0	0	ADVANCE	2
2	1	1	0	0	0	ADVANCE	3
3	1	1	2	0	0	ADVANCE	4
4	1	1	2	2	0	ADVANCE	5
5	1	1	2	2	3	DISPLAY	5
6	1	1	2	2	4(0)	BACKUP	4
7	1	1	2	3	0	ADVANCE	5
8	1	1	2	3	4(0)	BACKUP	4
9	1	1	2	4(0)	0	BACKUP	3
10	1	1	3	0	0	ADVANCE	4
11	1	1	3	2	0	ADVANCE	5
12	1	1	3	2	4(0)	BACKUP	4
13	1	1	3	3	0	ADVANCE	5
14	1	1	3	3	2	DISPLAY	5
15	1	1	3	3	4(0)	BACKUP	4
16	1	1	3	4(0)	0	BACKUP	3
17	1	1	4(0)	0	0	BACKUP	2
18	1	2	0	0	0	ADVANCE	3

v_i column: gives value of VCR(i) at end of stage.

k column: gives value of parameter k at end of stage.

4(0): indicates Next returns 4, which backtrack conrol then resets to 0, as required for BACKUP action.

Figure 7-8. Partial Trace of Backtrack Vector for for Graph of Figure 7-5 where M = 3.

is ignored. If we define the cost of a search algorithm does as the number of tree edges it explores, then the traversal order of Figure 7-6 has cost 30, while the traversal in Figure 7-7 has cost 10.

Figure 7-7 here

In summation, backtracking allows us to traverse a search tree, while pruning unncessary subtrees from the search. In tree terminology, a partial coloring VCR(1..i) corresponds to a search path through the tree down to a tree vertex at level i. We can extend the search path from its end vertex at level i to any of M possible successors. We can rephrase our previous rules using this terminology as follows.

(1) *Advance*: If a valid extension of the search path is available, the search path advances to the next level in the tree.

(2) *Back Up*: If no valid extension is available, the search path backs up to its previous vertex, and continues the search process from there.

(3) *Exit*: If there is no further valid extension at the root of the search tree, the search terminates.

(4) *Display*: If the search path reaches an endpoint of the search tree, the algorithm displays the contents of VCR, which corresponds to a complete coloring of the graph.

The following procedure Color_Backtrack generates all the valid colorings of a graph G(V,E) that use at most M colors. We assume G is represented as an adjacency matrix. We use a procedure Next(VCR,k), whose input is a partial coloring VCR(1..k - 1) and k. Next returns in VCR(k) the next valid coloring for vertex k which is consistent with the given partial coloring, and sets VCR(k) to M + 1 if there is no valid extension of the partial coloring to k. Figure 7-8 gives an example for the graph of Figure 7-5 with M = 3. The trace shows the first 18 iterations of the search loop.

neighbors (which have indices on 1..k - 1).

Each attempted extension of the coloring has four possible outcomes. If the extension succeeds, the procedure advances to the next vertex k + 1. If the extension fails, the procedure restores vertex k to an uncolored state and backs up to consider a new coloring for the preceding vertex k - 1.

If the extension fails at the very first vertex, the procedure exits. If the extension succeeds at the last vertex, the procedure displays the complete coloring. The process continues until every valid coloring is found and displayed.

It is instructive to visualize the process in terms of a search tree. There is one level in the search tree for each vertex in the graph. The levels are indexed from 1 to |V|, and there is an additional (dummy) level zero. There is an edge in the tree for each possible choice of color for an edge of the graph, each tree edge corresponding to a choice of color for the lower level vertex the edge ends at. Figure 7-4 shows the tree of two-colorings for the graph of five vertices in Figure 7-5. Observe that if a connected graph has chromatic number equal to two, then all of these colorings are equivalent, in the sense that the partitions of the vertices they induce are all the same.

Figures 7-4 and 7-5 here

Each path from the root of the tree to an endpoint corresponds to an (a priori) possible coloring of the graph, though not necessarily a valid coloring. If we ignore the constraints imposed by adjacency considerations, there are $|V|^M$ ways to color a |V| vertex graph with at most M colors, corresponding to $|V|^M$ paths from the root to the endpoints of the tree. A path from the root to a vertex at level i corresponds to a (partial) coloring of the first i vertices of a graph. The subtree rooted at the vertex at level i contains all possible continuations of the partial coloring defined by the tree path up to that vertex.

We could find the valid colorings of a graph by traversing each of its search tree paths completely, not testing the validity of the corresponding coloring until the end of the path was reached. Of course, in an intelligent search, we would terminate a particular path as soon as we recognized that a partial color assignment was invalid.

Figure 7-6 shows the tree of attempted two-colorings for the graph of Figure 7-5, with the levels arranged in the order v_1, v_2, v_3, v_4, v_5. At each tree vertex, the left (right) edge corresponds to a choice of color 1 (2) for the vertex corresponding to the lower endpoint of the edge. In this case the graph has no valid two-coloring. For example, if v_1 and v_2 are 1-colored, v_3 cannot also be 1-colored. We indicate this by a dash (-) before the corresponding edge in the search tree.

Figure 7-6 here

We can also order the vertices differently. This may affect the efficiency of an intelligent search. For example, in Figure 7-7 we have ordered the vertices v_1, v_4, v_5, v_2, v_3. Under this order, much more of the search tree

security of the id since not even the system needs to know the coloring.

This is an example of a public-key cryptosystem. In such a system, part of the id for a valid user, in this case the graph, can be public. Only the coloring part has to be kept secret and that, unlike an ordinary password, need not be shared even with the system being accessed.

The following points remain to be considered:

(1) How to construct a random three-colorable graph, and
(2) How to prove one knows how to color a graph without revealing the coloring.

While it is difficult to three-color a given graph, it is easy to construct a random three-color graph inductively. Let G(V,E) be a random three-color graph constructed so far. To extend G, we create a new vertex v, randomly color v with one of the three colors, and then add edges between v and a random subset of the vertices in G with different colors than v. We repeat the process until the order of G is sufficiently large. Though, by construction, we can three-color the final graph, a computational barrier prevents anyone else from duplicating this feat.

We can convince an observer we can three-color a given graph without revealing its coloring as follows. For simplicity, we use some physical imagery. We lay the graph on a table with its vertices covered. The observer picks a random pair of adjacent vertices. We expose their colors, revealing their difference. We then secretly and randomly permute the colors, changing all vertices with color i (i = 1,2,3) to color p(i), where p is a randomly chosen permutation of 1, 2, and 3. (Only a permutation of the colors is allowed. We are not allowed to change the coloring substantively.) We repeat the process until the observer is satisfied we can three-color the graph.

Of course, if we did not actually have a three-coloring of the graph, at each step of the above procedure there would be a fixed chance r the observer would select for examination a pair of vertices that were incorrectly colored. The chance we could deceive the observer in n successive trials would be at most $(1 - r)^n$, which rapidly approaches zero. Thus, after some sufficiently large number of tests (the actual number depends on r, which in turn depends on |V(G)| and |E(G)|) the chance of deception can be made arbitrarily small. Furthermore, because we randomly permute the colors, the observer cannot combine the results of one test with that of another to deduce the whole coloring, or indeed any part of it. Since the permutation effectively erases the observer's memory, he essentially knows only for each test whether the outcome was successful or not.

Finding all the valid M-colorings of a graph G(V,E) is a good illustration of backtracking. We will use a vector VCR(|V|) to store the colors of the vertices. At each stage of the backtrack search, VCR(k) equals the color currently assigned to vertex k. We take VCR to be initially zero, and expand the coloring one vertex at a time, repeatedly extending the partial coloring represented by VCR(1..k - 1) to the larger coloring VCR(1..k) by assigning to vertex k the next color not conflicting with its previously colored

chromatic number of a surface of genus N is the greatest integer in

$$(7 + (1 + 48N)^{(1/2)})/2 .$$

The Heawood formula (correctly) returns 4 for the chromatic number of a sphere, a surface of genus 0. It returns 7 as the chromatic number of a torus, a surface of genus 1. Observe that K(7) which has VCHR = 7, can be embedded on the torus.

7-2 MODELS: CONSTRAINED SCHEDULING AND ZERO-KNOWLEDGE PASSWORDS

CONSTRAINED SCHEDULING

Colorings can be used to model scheduling activities with overlapping resource requirements. Let T_1, ..., T_n be a set of tasks, each taking a unit time. Suppose some of the tasks cannot be scheduled concurrently because they use resources that are unshareable. We can summarize such concurrency constraints by a *conflict matrix* C where C(i,j) is 0 if tasks T_i and T_j require no common resource, and is 1 otherwise. We can interpret the conflict matrix as a *conflict graph* where each vertex corresponds to a task and there is an edge between a pair of vertices if the corresponding tasks cannot be scheduled concurrently. A k-coloring of the conflict graph partitions its vertices into k differently colored sets of tasks. If we schedule all the tasks numbered (colored) i for time period i, all the tasks will be finished by time k + 1. The length of a shortest possible schedule equals the chromatic number of the conflict graph. Figure 7-3 shows an example.

Figure 7-3 here

ZERO-KNOWLEDGE PASSWORDS

The chromatic number of a graph is not easy to compute. Indeed, it is an example of what we will later call an NP-Complete problem. NP-Complete problems do not seem to be solvable by algorithms with polynomial performance bounds. Generally, this is a disadvantage. But, curiously, applications have recently been found that rely precisely on the difficulty of solving of such problems.

A common security problem is to restrict access to a system to a list of valid user's. We can use graph colorings to accomplish this in an extremely reliable manner. The idea is for each user to randomly generate a three-colorable graph. We use the graph as the user identification number. To access the system, a user provides a graph-id, and proves ownership of the graph by three-coloring it. The graph-id is unforgeable because, even if someone else gets a copy of the graph, it is useless without its coloring, since it is computationally intractable to find a three-coloring of a given graph. Thus, as long as the coloring is kept secure, the graph establishes the user's identity in a way that cannot be forged. Furthermore, we can actually show we can three-color a graph without even revealing the coloring by a probabilistic procedure we will describe. This allows us to use the same id repeatedly, since the coloring is never revealed. It also further enhances the

THEOREM (BIPARTITE EDGE CHROMATIC NUMBER) If G(V,E) is bipartite, ECHR(G) = max(G).

The proof is by induction on the number of edges in G. Fix the number of vertices in G, and suppose G equals G' U (u,v) where G' has n edges and maximum degree m. If u or v is of degree m in G', max(G) is m + 1; hence we can color (u,v) with m + 1, and the induction follows. Otherwise, both u and v have degree in G' at most m - 1. If neither u nor v are incident with an edge colored with some color k, we can color (u,v) with k, using at most m colors for G, and again the induction follows.

Otherwise, there is some color k not incident with u and some color j not incident with v. Consider the set of vertices reachable from u by paths in G' containing only edges colored k or j. Let S be the induced subgraph on these vertices, and containing only edges colored k or j. Since G' is bipartite, any path from u to v has odd length. Therefore, any path from u to v in S must begin and end with the same color. Therefore, v cannot be in S since any path from u to v in S would begin and end with a j-colored edge which could not be incident with v. Therefore, if we interchange the colors of all the edges in S, changing j to k and k to j, we still maintain a valid edge coloring of G'. Since the edges incident with v are unaffected by this change, no edge incident with u or v is colored k in the new coloring. This corresponds to the previous case, and so completes the proof by induction.

The most well-known result in graph theory is the famous

THEOREM (FOUR-COLOR THEOREM) If G is a planar graph,

$$VCHR(G) \leq 4.$$

The proof of the Four-Color Theorem is enormously complicated. The weaker result

$$VCHR(G) \leq 5$$

is much easier to prove. We will later describe an algorithm for five-coloring a planar graph based on the proof of this Five-Color Theorem.

The Four-Color Theorem relates the planarity and chromatic number of a graph. This can be generalized to more complex surfaces than planes. Indeed, there is a simple relationship between the embeddability of a graph on a manifold, the maximum chromatic number of the graph, and the so-called topological genus of the manifold. Roughly speaking, a *surface of genus N* is a sphere to which N "handles" have been added. Thus, a sphere itself has genus zero, while a torus, which can be visualized as a sphere with one handle attached to its surface, has genus 1. The chromatic number of a surface S of genus N is the maximum chromatic number of all graphs that can be embedded on S. In this terminology, the Four-Color Theorem asserts that the sphere (which is equivalent for embedding purposes to the plane) has chromatic number 4. The following theorem is a vast generalization of this.

THEOREM (HEAWOOD MAP COLORING THEOREM) For every positive integer N, the

7 GRAPH COLORING

7-1 BASIC CONCEPTS

A *k-coloring* of a graph G is a mapping of V(G) onto the integers 1..k such that adjacent vertices map into different integers. A k-coloring partitions V(G) into k disjoint subsets such that vertices from different subsets have different colors. Of course, it can happen that a pair of different k-colorings may nonetheless partition V(G) into the same parts, in which case, the colorings are in a sense equivalent. Given a k-coloring, it is customary to call the integer a vertex maps into its *color*. A graph G is *k-colorable* if it has a k-coloring. The smallest integer k for which G is k-colorable is called the *chromatic number* or *vertex chromatic number* (VCHR) of G. A graph whose chromatic number is k is called a k-chromatic graph. Figure 7-1 shows a four-chromatic graph and Figure 7-2 shows the smallest triangle-free four-chromatic graph, the Grotzsch graph. A mapping from E(G) into the set of integers 1..k such that adjacent edges map into different integers is called an *edge coloring* of G. The cardinality of a minimum cardinality edge coloring of G is called the *edge chromatic number* (ECHR) of G. There are extensive and difficult theoretical results about coloring. We will mention a few.

Figures 7-1 and 7-2 here

THEOREM (UPPER BOUNDS) If a connected graph G(V,E) is neither an odd cycle nor a complete graph, then the chromatic number of G satisfies

$$VCHR(G) \leq \max(G) \quad \text{(Brooks' Inequality)}$$

For any nontrivial graph G, the edge chromatic number ECHR of G satisfies

$$ECHR(G) \leq 1 + \max(G) \quad \text{(Vizing's Inequality).}$$

Obviously, max(G) is a lower bound for ECHR(G). Both Brooks' and Vizing's Inequalities are nontrivial to prove. However, a weakened version of Brooks' Inequality,

$$VCHR(G) \leq 1 + \max(G)$$

is easy to prove, and is realized by the following procedure. As usual, Next(G,v) returns a next vertex in G or fails:

while Next(G,v)

  do Set color(v) to a color not used by the neighbors of v

The body of the **while** loop always succeeds because the neighbors of any vertex exhaust at most max(G) colors. Therefore, if we allow max(G) + 1 colors, we can always color v.

The following theorem of Koenig is well known.