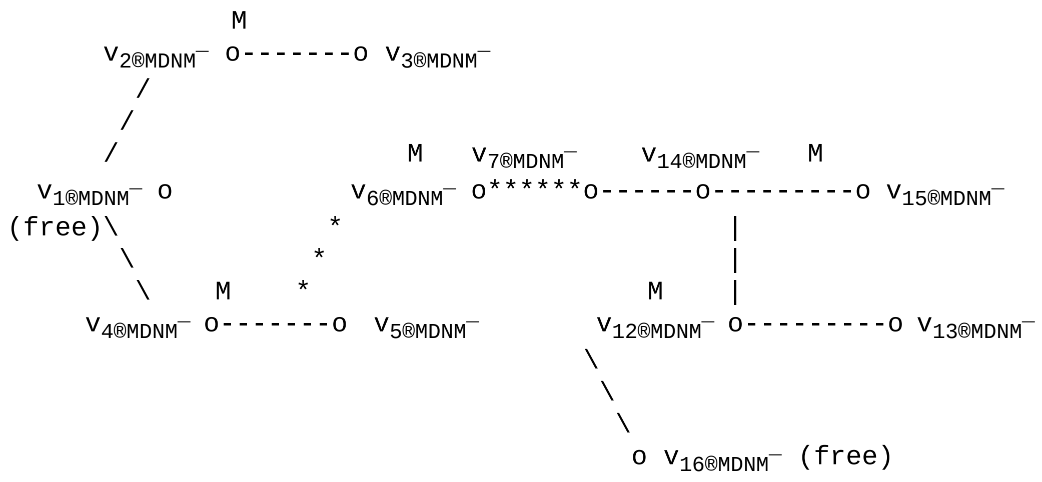
v3®MDNM¯ v4®MDNM¯

Figure 8-10. Diagram for Proof of Correctness.

®

(e) $C_{2®MDNM}$¯ Expanded and Even Part of $C_{2®MDNM}$¯ Used.
®



Even part of $C_{1®MDNM}$¯ (starred) from entry at $v_{7®MDNM}$¯ to exit at

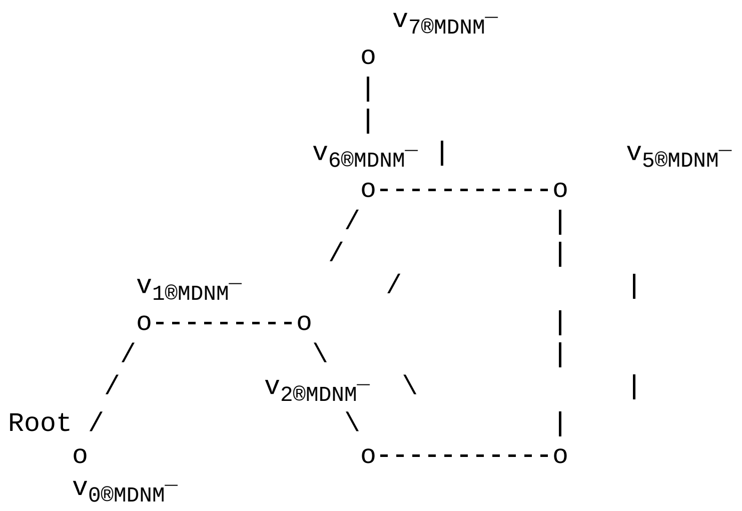
$v_{5®MDNM}$¯: $v_{7®MDNM}$¯,$v_{6®MDNM}$¯,$v_{5®MDNM}$¯.

Expanded Augmenting Path: $v_{1®MDNM}$¯, $v_{4®MDNM}$¯, $v_{5®MDNM}$¯, $v_{6®MDNM}$¯, $v_{7®MDNM}$¯, $v_{14®MDNM}$¯,

$v_{15®MDNM}$¯, $v_{13®MDNM}$¯, $v_{12®MDNM}$¯, $v_{16®MDNM}$¯.

(f) $C_{1®MDNM}$¯ Expanded with the Even Part of $C_{1®MDNM}$¯ Used.

Figure 8-9. Edmonds' Maximum Matching Algorithm Example.
®



```
                                        \  Vertex incident
                                         o with blossom
                                       v16®MDNM¯
```

(c') Details of $C_{2®MDNM¯}$ Cycle.

®

```
              v2®MDNM¯    M    v3®MDNM¯
               o------o
              /
             /
            /
        v1®MDNM¯ o
      (free)\
             \
              \     M    C2®MDNM¯
            v4®MDNM¯ o-------o
                          \
                           \
                            \
                             o v16®MDNM¯
                               (free)
```

(d) Augmenting Vertex $v_{16®MDNM¯}$ Detected via Edge ($C_{2®MDNM¯}$, $v_{16®MDNM¯}$)
(= ($v_{12®MDNM¯}$,$v_{16®MDNM¯}$)).

®

```
                 M
         v2®MDNM¯ o------o v3®MDNM¯
            /
           /
          /
     v1®MDNM¯ o                   M
    (free)\        v14®MDNM¯ o******** o v15®MDNM¯
           \           *           *
            \    M   *             *
         v4®MDNM¯ o------o C1®MDNM¯         *
                                   *
                            M      *
                   v12®MDNM¯ o******** o v13®MDNM¯
                                *
                                 *
                                  *
                                   o v16®MDNM¯ (free)
```

Even part of $C_{2®MDNM¯}$ (starred) from entry at $v_{12®MDNM¯}$ to exit at

$C_{1®MDNM¯}$: $v_{12®MDNM¯}$, $v_{13®MDNM¯}$, $v_{15®MDNM¯}$, $v_{14®MDNM¯}$, $C_{1®MDNM¯}$.

```
o-------o v5®MDNM¯                                  |    cycle
v4®MDNM¯       \                                    |
        ^ \                                     |
        |  \                                    |
        |   \                                   |
      Exit   \     M                            |
      from    o--------o-------------------
      cycle    v8®MDNM¯       v9®MDNM¯(outer)
```

(b') Details of $C_{1®MDNM¯}$ Cycle.

®

```
        v2®MDNM¯    M
          o-------o v3®MDNM¯
         /
        /
       /
   v1®MDNM¯ o             v14®MDNM¯      v15®MDNM¯
 (free) \             o---------o
         \           /     M    |
          \    M    /           |      New
       v4®MDNM¯ o------o C1®MDNM¯        | <--- Odd
                   \            |      Cycle C2®MDNM¯
                    \     M     |
              v12®MDNM¯ o---------o v13®MDNM¯
```

$C_{2®MDNM¯}$: $C_{1®MDNM¯}$, $v_{12®MDNM¯}$, $v_{13®MDNM¯}$, $v_{15®MDNM¯}$, $v_{14®MDNM¯}$, $C_{1®MDNM¯}$

Predecessor of $C_{2®MDNM¯}$: $v_{4®MDNM¯}$

Exit vertex from $C_{2®MDNM¯}$: $C_{1®MDNM¯}$

(c) $C_{1®MDNM¯}$ Collapsed and New Odd Cycle $C_{2®MDNM¯}$ Detected.

®

```
              Exit
              from    v14®MDNM¯      v15®MDNM¯
              cycle  o---------o
                 |  /    M     |
                 | /           |
                 v/            |      Edge
    v4®MDNM¯ o--------o C1®MDNM¯        | <--- (v13®MDNM¯,v15®MDNM¯)
             M    \            |      causes
      ^            \           |      cycle
      |             \    M     |
      |       v12®MDNM¯ o---------o v13®MDNM¯
      |                \
  Predecessor           \
         v4®MDNM¯  M    \                \        v13®MDNM¯     |
                    \                \                 |
                     \                \                |
                      \                o v16®MDNM¯ (free)  |
                       \     M                         |
                        o--------o------------------
                         v8®MDNM¯      v9®MDNM¯

 (a) Graph with Initial Matching and Free Vertices v1®MDNM¯ and v16®MDNM¯.
®


         v2®MDNM¯  M                         v10®MDNM¯     v11®MDNM¯
         o------v3®MDNM¯                -----o--------o---------
        /                   |          M             |
       /                    |                        |
      /                 M   |     v14®MDNM¯  M    v15®MDNM¯      |
  v1®MDNM¯ o             v6®MDNM¯ o------o-----o--------o          |
(free)\             /     v7®MDNM¯|                        |       Odd
       \           /        |                        |  <--- Cycle
        \         /         |     v12®MDNM¯  M           |       C1®MDNM¯
         o-------o v5®MDNM¯        -----o--------o          |
         v4®MDNM¯  M    \                       v13®MDNM¯      |
                    \                                 |
                     \                                |
                      \                               |
                       \     M                        |
                        o--------o------------------
                         v8®MDNM¯      v9®MDNM¯

        C1®MDNM¯: v5®MDNM¯,v8®MDNM¯,v9®MDNM¯,v11®MDNM¯,v10®MDNM¯,v7®MDNM¯,v6®MDNM¯,v5®MDNM¯

        Predecessor of C1®MDNM¯: v4®MDNM¯

        Exit vertex from C1®MDNM¯: v5®MDNM¯

    (b) Odd Cycle Detected on Scanning Edge (v9®MDNM¯,v11®MDNM¯).
®


                                 v10®MDNM¯      v11®MDNM¯(outer)
                             -----o--------o---------
                            |          M             |
                            |                        |
                        M   |                        |
    Predecessor   v6®MDNM¯ o------o                      |
         |          /     v7®MDNM¯                       |    Edge
         |         /                                 | <--
(v9®MDNM¯,v11®MDNM¯)
         v   M    /                                  |    causes
        Figure 8-7. Matching on Graph with Odd Cycle.
®
```

```
                                            o E: External Vertex
                                            |
                                            |
                          Cycle Predecessor |
                          of E --->  Entry  v
  o E: External Vertex                o-----o-----o
  |                                   |           |
  |                                   |           |
  v                                   |           |
  o b: Blossom                        |           | <-- Blossom
  |                                   o           |     Cycle B
  |                                   |           |
  v                   Cycle           |           |
  o P: Predecessor    Exit            |           |
                      Vertex --> Exit o-----o-----o
                                      |
                                      |
                                      v
                                      o P  <-- Tree Predecessor

   (a) Unexpanded Blossom         (b) Expanded Blossom

 E:     vertex external to blossom with predecessor in blossom

 b:     blossom vertex prior to expansion

 P:     search tree vertex which is predecessor of blossom b

 Entry: vertex in expanded blossom which is predecessor of E

 Exit:  vertex in expanded blossom whose predecessor is P
```

```
             Figure 8-8. Blossom Structure.
®
```

```
        v2®MDNM¯ M                    v10®MDNM¯    v11®MDNM¯
         o------v3®MDNM¯          -----o--------o--------
        /                   |          M            |
       /                    |                       |
      /                 M   |     v14®MDNM¯ M   v15®MDNM¯    |
  v1®MDNM¯ o          v6®MDNM¯ o------o-----o--------o       |
(free)\             /     v7®MDNM¯|             |        |
       \           /        |              |        |
        \         /         |     v12®MDNM¯  M  |        |
         o-------o v5®MDNM¯       -----o--------o        |
                  /           \
                 o             o
              P1®MDNM¯(1)          P3®MDNM¯(1)
```

(c) Second Reduced Graph.

Figure 8-6. Testing for Deadlock Freedom.

®

SECTION 8-3

```
                            Root
                          o
                         / \
                        /   \ V2®MDNM¯
                    V1®MDNM¯ o      o--------o V5®MDNM¯
                       |     |
                     M |     | M
                       |     |
                       o-----o
                    V3®MDNM¯           V4®MDNM¯
```

(a) Graph with Odd Cycle.

®

```
                            Root (free)
                          o
                         / \
                     M /   \ V2®MDNM¯
                       /     \     M
                   V1®MDNM¯ o       o--------o V5®MDNM¯ (free)
                      |       |
                      |       |
                      |       |
                      o-------o
                   V3®MDNM¯     M    V4®MDNM¯
```

(b) Augmenting Path.

®

```
                            Root* (free)
                          o
                           \
                            \
                             \
                              o V5®MDNM¯ (free)
```

(c) Odd Cycle Collapsed.

Notation:

Claim(i,j):   amount of resource j claimed   by process i

Alloc(i,j):   amount of resource j allocated to process i

Request(i,j): amount of resource j requested by process i

Figure 8-5. Constructing Deadlock Configuration (b)
from a Complete Matching in (a).

®

```
       ------------
      |     R1®MDNM¯(2)  | R2®MDNM¯(1)  R3®MDNM¯(1)
      |       o     o       o
      |      / \   / \     /|\
      |     /   \ /   \   / | \
      |    /     \     \ /  |  \
      |  M     M M     \   M   \
      | /     /   \   / \  |    \
      |/     /     \ /   \ |     \
      o     o       o      o      o
 P1®MDNM¯(1) P2®MDNM¯(1)  P3®MDNM¯(1)  P4®MDNM¯(1)  P5®MDNM¯(1)
```

(a) Maximum Degree Constrained Matching for G(P) of Figure 8-4.

®

```
       ------------
      |     R1®MDNM¯(2)  | R2®MDNM¯(1)
      |       o     o
      |      / \   / \
      |     /   \ /   \
      |    /     \     \
      |  M     M M     \
      | /     /   \     \
      |/     /     \     \
      o     o       o     o
 P1®MDNM¯(1) P2®MDNM¯(1)  P3®MDNM¯(1)  P4®MDNM¯(1)
```

(b) First Reduced Graph.

®

```
      R1®MDNM¯(2)
        o
       / \
      /   \
     /     \
    M       M
   /         \
       ------------
      |     R1®MDNM¯(2)  | R2®MDNM¯(1)  R3®MDNM¯(1)
      |      o     o       o
      |     / \   / \     /|\
      |    /   \ /   \   / | \
      |   /     \     \ /  |  \
      |  /     / \     \   |   \
      | /     /   \   / \  |    \
      |/     /     \ /   \ |     \
      o     o       o      o       o
  P1®MDNM¯(1) P2®MDNM¯(1)  P3®MDNM¯(1)  P4®MDNM¯(1)  P5®MDNM¯(1)
```

(b) Shortage Graph Shortage(P,P,R,Claim,Avail).

Figure 8-4. Example Shortage Graph.

®

```
           R1®MDNM¯(2)  R2®MDNM¯(1)
              o      o        Avail(1)   = 10
             /|\    /|        Avail(2)   = 1
            / | \  / |
           /  |  \/  |        Claim(1,1) = 1
        M /  M|  /\  | M      Claim(2,1) = 6
         /    | /  \ |        Claim(3,1) = 5
        /     |/    \|        Claim(1,2) = 0
       o      o      o        Claim(2,3) = 1
   P1®MDNM¯(1)  P2®MDNM¯(1)   P3®MDNM¯(1)     Claim(3,2) = 1
```

(a) Shortage Graph with Complete Matching.

®

Alloc(1,1) = 0	Alloc(2,1) = 0
Alloc(2,1) = 5	Alloc(2,2) = 1
Alloc(3,1) = 5	Alloc(2,3) = 0
Request(1,1) = 1	Request(1,2) = 0
Request(2,1) = 1	Request(2,2) = 0
Request(3,1) = 0	Request(3,2) = 1

(b) Deadlock State.

```
          T1®MDNM¯      T4®MDNM¯
          o--------o
          |        |
          |        |     o T5®MDNM¯
          |        |
          o--------o
          T3®MDNM¯      T2®MDNM¯
```

(b) Associated Graph C(U(T(G)).

®

```
          T1®MDNM¯      T4®MDNM¯
          o--------o
          |        |
        M |      M |     o T5®MDNM¯
          |        |
          o--------o
          T3®MDNM¯      T2®MDNM¯
```

Associated Optimal Schedule

```
   Time = 0: T1®MDNM¯, T3®MDNM¯

   Time = 1: T2®MDNM¯, T4®MDNM¯

   Time = 3: T5®MDNM¯
```

(c) Associated Graph C(U(T(G)).

Figure 8-3. Lower Bound Example.

®

SECTION 8-2-3

	R1®MDNM¯	R2®MDNM¯	R3®MDNM¯	
P1®MDNM¯	2	1	0	Avail(1) = 2
P2®MDNM¯	0	2	0	Avail(2) = 4
P3®MDNM¯	2	0	1	Avail(3) = 3
P4®MDNM¯	0	2	1	
P5®MDNM¯	0	0	2	

(a) Claim Matrix and Avail.

®

```
 1       3       6       2
 o       o<------o       o<----
 ^       ^       ^       ^     |
 |       |       |       |     |
  ---o---         ---o---      |
     ^4              ^7        |
     |               |         |
     |       9       |         |
     o<------o------>o         |
     |5               8        |
     |                         |
      -------------------------
```

(g)

Figure 8-1. Priority Assignment.

®

Time	Available	P_1®MDNM¯	P_2®MDNM¯
0	{9}	9	-
1	{5,8}	5	8
2	{4,7}	4	7
3	{1,2,6}	6	2
4	{1,3}	1	3

Figure 8-2. Optimal Schedule for Figure 8-1.

®

```
T1®MDNM¯------->T2®MDNM¯------->T5®MDNM¯
                ^
                |
                |
T3®MDNM¯------->T4®MDNM¯--------
```

(a) Task Precedence Digraph G.

®

```
1       3       6       2
o       o<------o       o<----
^       ^       ^       ^     |
|       |       |       |     |
 ---o---         ---o---      |
    ^4              ^(2,6)    |
    |               |         |
    |               |         |
    o<------o------>o         |
    |5                        |
    |                         |
     -------------------------
```

(d)

®

```
1       3       6       2
o       o<------o       o<----
^       ^       ^       ^     |
|       |       |       |     |
 ---o---         ---o---      |
    ^4              ^7        |
    |               |         |
    |               |         |
    o<------o------>o         |
    |5               (7)      |
    |                         |
     -------------------------
```

(e)

®

```
1       3       6       2
o       o<------o       o<----
^       ^       ^       ^     |
|       |       |       |     |
 ---o---         ---o---      |
    ^4              ^7        |
    |               |         |
    |     (5,8)     |         |
    o<------o------>o         |
    |5               8        |
    |                         |
     -------------------------
```

(f)

®

FIGURES FOR CHAPTER 8

SECTION 8-2-1

```
1       3      (3)      2
o       o<------o       o<----
^       ^       ^       ^     |
|       |       |       |     |
 ---o---         ---o---      |
    ^(1,3)          ^         |
    |               |         |
    |               |         |
    o<------o------>o         |
    |                         |
    |                         |
     -------------------------
      (a)
```

®

```
1       3      (3)      2
o       o<------o       o<----
^       ^       ^       ^     |
|       |       |       |     |
 ---o---         ---o---      |
    ^4              ^         |
    |               |         |
    |               |         |
    o<------o------>o         |
    |(2,4)                    |
    |                         |
     -------------------------

      (b)
```

®

```
1       3      (3)      2
o       o<------o       o<----
^       ^       ^       ^     |
|       |       |       |     |
 ---o---         ---o---      |
    ^4              ^         |
    |               |         |
    |               |         |
    o<------o------>o         |
    |5                        |
    |                         |
     -------------------------

       (c)
```

®

(8) Prove there are at most |®MDSU¯(|V|/|)®MDNM¯ maximal independent sets in a graph G(V,E).

(9) Prove that if VC(G) = k, then at least one of the colors in a k-coloring of G determines a maximal independent set.

(10) Prove that every cubic graph with no articulation point is 1-factorable.

(11) Construct a cubic graph containing no 1-factors.

(12) Use the Maximum-Flow Minimum-Cut to show that if G is bipartite, Max-Match(G) equals Min-VCov(G).

(13) Model the degree constrained maximum matching problem on a bigraph as a flow problem.

(14) Write a program that determines whether a system is deadlock free, or if it is not, outputs a resource request/allocation sequence leading to a deadlock.

(15) Let G(V,E) be a bipartite graph, and let M be a matching on G. Then, prove there is a maximum matching M' on G such that every vertex matched in M is also matched in M'.

(16) Let T(V,E) be a tree, and let o(T,v) be the number of components of T - v of odd order. Prove that T has a perfect matching if and only if o(T,v) equals 1 for every v in T.

(17) Construct a counterexample to the following greedy algorithm for the minimum vertex cover of a graph G, and try to estimate its worst case error (assume G has no isolated vertices). Repeat the following until G is empty: Select a vertex v of maximum degree; add v to the cover; and delete its incident edges and any resulting isolated vertices from G.

(18) A vertex independent set in G(V,E) corresponds to a complete subgraph in the complement of G. Referring to the exercise on a backtracking algorithm for cliques in Chapter 2, show how this relates to a backtracking algorithm for independent sets.

(19) Show how to find the maximum number of vertex disjoint paths between a given pair of sets of vertices X and Y in a graph G(V,E) by modelling the problem as a matching problem. (See the section on problem transformation in Chapter 2, where this is done for the case where X and Y have the same cardinality.)

CHAPTER 8: REFERENCES AND FURTHER READING

Edmonds' famous algorithm was given in Edmonds (1965). There have been a variety of improvements since then, including improved algorithms for bipartite graphs. Gabow (1976) and Lawler (1976) show how to implement Edmonds' algorithm in O(|V|$^{\text{®MDSU¯3®MDNM¯}}$) time. A special case linear time disjoint set union algorithm of Gabow and Tarjan (1983) yields an algorithm with time O(|E||V|). See also Tarjan (1983), which, along with Edmonds' original article, is the basis for some of our discussion. For an extensive discussion of results and algorithms on matching, see Swamy and Thulasiraman (1981). Behzad, et al. (1979) also gives the basic theory of covers, etc. The proof of Gallai's Theorem is based on Behzad, et al. (1979). Valiant (1975) gives the optimal parallel selection analysis. Refer to Turan (1954) for a proof of the extremal theorem Valiant uses. See Coffman and Denning (1973) for a discussion of the two-processor scheduling algorithm. The deadlock-free system characterization and algorithm is after Kameda (1980). The reducing path characterization of a minimum edge cover in the exercises is from Norman and Rabin (1959).

CHAPTER 8: EXERCISES

(1) Prove that a tree has at most one perfect matching.

(2) Use the duality implied in Gallai's Theorem to show how to construct a minimum edge cover from a maximum matching.

(3) An alternating path with respect to an edge cover®MDNM¯ C on a graph G(V,E) is a path whose edges are alternately in C and outside C. A reducing path with respect to an edge cover®MDNM¯ C is an alternating path whose terminal edges are in C and whose terminal vertices are incident to edges of C which are not terminal edges of C. Prove the following analog of the augmenting path characterization for maximum matchings. An edge cover C is a minimum cover if and only if there is no reducing path with respect to C. For this condition to work, must we allow the reducing path to be closed? Give an example, or clearly address this in the proof.

(4) What is the relationship between a coloring of a graph and a partition of a graph into independent sets of vertices?

(5) Give an algorithm that finds a maximum cardinality vertex independent set on a bipartite graph.

(6) Consider the following method for finding a maximum independent set in a graph G(V,E). Let v be a vertex in G of degree at least 3. Either v is in a maximum independent set of G or it is not. If it is, then none of the vertices in ADJ(v) can be in the maximum independent set. If it is not, then the maximum independent set is contained in G - {v}. Develop these observations into a recursive algorithm for finding a maximum independent set, and determine a bound on the performance of the algorithm using recurrence relations. Since this problem is NP-Complete, the bound is exponential.

(7) Prove that every bipartite graph G is a subgraph of a max(G)-regular bipartite graph.

```
  then®MDNM¯  Get_Augmenting_Path (G,Root,v)
        Appply_Augmenting_Path(G,Root,v)
        Clear(G)

  else®MDNM¯  Remove_Tree (G)

End_Procedure®MDNM¯_Maximum_Matching
®
```

Figure 8-9 here

The proof of the correctness of the exclusion of Hungarian trees from subsequent searches is the same as in Chapter 1. The correctness of the blossom management procedures is established by the following theorem.

THEOREM (BLOSSOM MANAGEMENT) The Find_Augmenting_Tree algorithm succeeds if and only if G has an augmenting path.

The sufficiency is clear from the blossom expansion techniques that we have described. Therefore, it only remains to prove that if there is an augmenting path in the original graph, then some such path must be found by Find_Augmenting_Tree. Let us first observe that if the search tree ever becomes blocked, any adjacent outer vertices are, by that point, collapsed into a common blossom, as follows directly from the nature of the search procedure. Suppose now that Find_Augmenting_Tree fails, even though there is an augmenting path $v_{0®MDNM}$¯, ..., $v_{m®MDNM}$¯ (where m is necessarily odd) in the original graph G. It will be helpful to refer to Figure 8-10.

Figure 8-10 here

Let k be the largest index on 0, .., m such that, for every vertex $v_{j®MDNM}$¯ on the path with $j \leq k$, $v_{j®MDNM}$¯ (or a blossom that $v_{j®MDNM}$¯ lies in in the collapsed graph) is in the search tree rooted at $v_{0®MDNM}$¯ (or rooted at a blossom containing $v_{0®MDNM}$¯). Observe that $k \leq m - 1$, $v_{k®MDNM}$¯ must be an Inner vertex of the search tree, and the vertex matched with $v_{k®MDNM}$¯ must be $v_{k-1®MDNM}$¯, from which it follows that the index k must be even.

Now, let r be the smallest even index such that $v_{r®MDNM}$¯ is inner. By our initial observation, one of the vertices $v_{0®MDNM}$¯,..., $v_{r-1®MDNM}$¯ must be inner, or else $v_{0®MDNM}$¯,..., $v_{r®MDNM}$¯ would have been collapsed into a single blossom by the time the search tree was blocked, contrary to $v_{r®MDNM}$¯ being inner. Therefore, we can let i be the largest index less than r such that $v_{i®MDNM}$¯ is inner. Then, i must be odd, and so the matched vertex to $v_{i®MDNM}$¯ must be $v_{i+1®MDNM}$¯. Therefore, the vertices $v_{i+1®MDNM}$¯, ..., $v_{r-1®MDNM}$¯ must be collapsed to a single vertex which is matched to both $v_{i®MDNM}$¯ and $v_{r®MDNM}$¯, which is a contradiction. It follows that Find_Augmenting_Path must find some augmenting path, if one exists, which completes the proof of the theorem.

```
Function®MDNM¯ Find_Augmenting_Tree (G,Root,v)

(* Tries to find an augmenting path from Root to v, or fails *)

var®MDNM¯ G: Graph
    v, w, Root, Q: Vertex pointer
    Head: Vertex pointer function
    Next, Find_Augmenting_Tree, Empty: Boolean function

Set®MDNM¯ Find_Augmenting_Tree to False

Create(Q); Enqueue (Root,Q)

repeat®MDNM¯

  while®MDNM¯ Next(Head(Q), v) do®MDNM¯

    if®MDNM¯  v free and®MDNM¯ <> Root then®MDNM¯ (* Augmenting path found *)
                                Set®MDNM¯ Pred(v) to Head(Q)
                                Set®MDNM¯ Find_Augmenting_Tree to True
                                Return®MDNM¯

    else®MDNM¯ if®MDNM¯ v unscanned    then®MDNM¯ (* Extend search tree *)
                                Let w be vertex matched with v

                                Set®MDNM¯ Pred(v) to Head(Q)
                                Set®MDNM¯ Status(v) to Inner

                                Set®MDNM¯ Pred(w) to v
                                Set®MDNM¯ Status(w) to Outer

                                Enqueue(w,Q)

    else®MDNM¯ if®MDNM¯ v Outer        then®MDNM¯ Create_Blossom (G,Head(Q),v)

    else®MDNM¯ (* v Inner *)     then®MDNM¯ (* Ignore *)

until®MDNM¯ Empty(Dequeue(Q))

End_Function_®MDNM¯Find_Augmenting_Tree


Procedure®MDNM¯ Maximum_Matching (G)

(* Finds a maximum matching in G *)

var®MDNM¯  G: Graph
     Root, v: Vertex pointer
     Next_Free, Find_Augmenting_Tree: Boolean function

while®MDNM¯   Next_Free (G,Root)   do®MDNM¯

  if®MDNM¯    Find_Augmenting_Tree (G,v,Root)
```

O(|V|$^{\text{®MDSU¯3®MDNM¯}}$). Just as in the binary case, the search tree can become blocked, in which case, the blocked (or *Hungarian®MDNM¯*) search (sub)tree can be ignored in *all®MDNM¯* subsequent searches.

Our algorithm uses the straightforward approach. A blossom B is recognized when an edge between a pair of outer vertices is scanned. It consists of that edge, plus the two tree paths from the unique common ancestor of the endpoints of the edge. The blossom is handled by shrinking it into a super-vertex b, which is labelled as free, if it contains the root of the tree, and always as an outer vertex, to ensure it is scanned from in the collapsed graph G(b), obtained from G by shrinking the blossom B into a single vertex b. Thus, corresponding to any vertex v in V(G) - V(B) which is adjacent to a vertex w in V(B), there is a vertex v in G(b) which is adjacent to b in V(G(b)). Any self-loops, caused by mutually adjacent vertices in V(B) in G, are deleted in G(b). Any parallel edges, caused if more than one vertex in B is adjacent to a common vertex outside V(B), are also removed. Thus, G(b) is a simple loop-free graph.

A high level view of the augmenting search tree algorithm is given in Find_Augmenting_Search_Tree. It is similar to the bipartite search tree algorithm, except that it includes a blossom recognition and collapse step. Also, the status of the vertices (inner or outer) has to be explicitly recorded, because of the different actions that are taken if the vertex being scanned is outer or inner. The function Next(u,v) returns the next unexplored edge (u,v) at u, and fails if there is none. If Next succeeds, Find_Augmenting_Tree either recognizes v as an augmenting vertex or determines that v is matched and so extends the search tree to v and its neighboring matching vertex, or recognizes that the edge (u,v) determines a blossom. Find_Augmenting_Tree terminates when either the search becomes blocked or when an augmenting vertex is found.

A suggestive outline of the driver algorithm is given in the procedure Maximum_Matching. The procedure Clear merely resets positional pointers and status flags, but does not have to reestablish the graph as it was before the blossoms off the augmenting path were introduced. Remove_Tree(G) just removes a Hungarian tree from G.

Get_Augmenting_Path extracts the augmenting path detected by the search algorithm. If necessary, it restores the part of G along the augmenting path from v to Root, by reexpanding any blossoms introduced along that path by Find_Augmenting_Tree. Since blossoms can be nested and can include the original root, this process can be quite complicated. But, the basic idea is simple. Whenever a blossom is recognized, as we backtrack along the augmenting path found by Find_Augmenting_Tree, we expand it and use the even length part of the blossom between the vertex we enter the blossom at, and the vertex from which we exit the blossom, as a part of the expanded augmenting path. Refer to Figure 8-8. Observe that only blossoms lying on the augmenting path have to be expanded. A blossom off the path merely remains part of the graph as it stands for the next search phase, and has to be expanded only if it becomes an Inner vertex with respect to some future search. A detailed example is shown in Figure 8-9.

Figure 8-8 here

characterization theorem, while the iterated steps in (4) are each justified by the Theorem on Deadlockable Processes.

(1) Compute Short(P,j) for every resource $R_{j®MDNM¯}$, and discard those for which Short(P,j) is negative. Then, construct the model bigraph G(P) on the remaining resources.

(2) Find a maximum degree constrained matching on G(P).

(3) If the matching is complete, then the system is not deadlock free.

(4) Otherwise, repeat the following step as many times as possible.

    Select any process vertex p not in the matching. Delete p and every resource vertex incident with p.

(5) The system is deadlock free, if, upon completion of the step (4) iterations, no process vertex remains.

An example is shown in Figure 8-6. Figure 8-6a gives an incomplete maximum matching ($p_{5®MDNM¯}$ is not matched). The initial iteration of step (4) deletes $P_{5®MDNM¯}$ and $R_{3®MDNM¯}$ (Figure 8-6b). This leaves $P_{4®MDNM¯}$ unmatched; so $p_{4®MDNM¯}$ is deleted in the next iteration of (4) (Figure 8-6c). At this point, every process vertex is matched; so no further iterations of (4) are possible. Since the final graph is nonempty, the system is not deadlock free.

8-3 MAXIMUM MATCHING ALGORITHM OF EDMONDS

We introduced the alternating paths algorithm for maximum matchings on bipartite graphs in Chapter 1. We now describe an alternating paths algorithm for maximum matchings on arbitrary graphs. The algorithm is complicated in the general case because the graph may contain cycles of odd length. See Figure 8-7a for an example of what can go wrong. If we start an augmenting search tree at Root, we may not detect the augmenting path Root-$v_{1®MDNM¯}$-$v_{3®MDNM¯}$-$v_{4®MDNM¯}$-$v_{2®MDNM¯}$-$v_{5®MDNM¯}$, which gives the augmented matching shown in Figure 8-7b: it depends on the order in which we happen to scan the vertices. The same thing happens if we start the search tree at $v_{5®MDNM¯}$. However, if we collapse an odd cycle as soon as it is detected, as shown in Figure 8-7c, we can recognize the augmenting path in the collapsed graph. If we then merely follow that augmenting path (from $v_{5®MDNM¯}$ back to Root), using the even length part of the odd cycle, we get an augmenting path in G, which yields the maximum matching in Figure 8-7b.

Figure 8-7 here

The same idea works for arbitrary graphs, though there are considerable variations depending on how the odd cycles (or *blossoms®MDNM¯*, as thay are called) are handled. In the most straightforward approach, where the blossoms are collapsed, and re-expanded later on demand if they lie on an augmenting path, the algorithm takes O($|V|^{®MDSU¯4®MDNM¯}$) time. A variation due to Gabow (1976) avoids the explicit collapse and expansion and has time

Figures 8-5 and 8-6 here

We can characterize deadlock free systems in terms of complete degree constrained matchings.

THEOREM (CHARACTERIZATION OF DEADLOCK FREEDOM) Let P = {$P_{\text{i®MDNM¯}}$, i = 1, ..., n} be a set of processes, and let R = {$R_{\text{j®MDNM¯}}$, j = 1, ..., m} be a set of resource types, with Claim and Avail as defined above. Then, the system is deadlock free if and only if there is no subset S of P such that Shortage(S,P,R,Claim,Avail) has a complete degree constrained matching.
®
We prove just the only if part of the theorem. Suppose a set of processes S exists for which Shortage(S,P,R,Claim,Avail) has a complete degree constrained matching M. Then, we can construct a deadlock state of the system as follows. Suppose there are $m_{\text{1®MDNM¯}}$ edges of M incident at a particular resource vertex r. By definition, each process incident on a matching edge of r has a nonzero potential claim against r. Let us assign a request of one unit of r to each such process, for a total of $m_{\text{1®MDNM¯}}$ requested units. Since the number of matching edges incident at r is at most equal to Short(S,r) by the degree constrained nature of the matching, it follows by the definition of Short(S,r) that

$$\text{Avail(j)} \le \sum_{\text{i in S}} \text{Claim(i,j)} - m_{\text{1®MDNM¯}}.$$
®
It follows from this that there is sufficient outstanding demand for r from S to deplete all the available supply of r. Therefore, let us assume that all Avail(j) units of r have already been distributed among all those processes in S that have nonzero claims against r, whether matched with r or not, and in such a way that none of the processes matched with r is allocated its full possible claim, thus allowing for a unit request for each of the matched processes. The resulting configuration blocks every process matched with r, since each such process is waiting for a unit of r, and all units of r have already been granted. If we construct a similar pattern of allocated claims and pending requests for every process in S; then in the resulting configuration, each process is blocked waiting for a resource held by another blocked process, that is, the system is deadlocked. This completes the proof of necessity. We refer to Kameda (1980) for the complete discussion. Figure 8-5 gives an example.

In order to define an algorithm that tests if a system is deadlock free, we need the following theorem.

THEOREM (DEADLOCK-ABLE PROCESSES) Let P = {$P_{\text{i®MDNM¯}}$, i= 1, ..., n} be a set of processes, and let R = {$R_{\text{j®MDNM¯}}$, j = 1, ..., m} be a set of resource types, with Claim and Avail as defined above. Then, if a process $P_{\text{i®MDNM¯}}$ can become deadlocked, then $P_{\text{i®MDNM¯}}$ must be matched to some resource vertex in every maximum degree constrained matching of G(P).

We refer to Kameda for the proof. An algorithm that tests if a system is deadlock free follows. Step(1) of the algorithm computes the shortage variables and deletes unrestricted respources. Step (3) is justified by the

Consider a system with a set P of n processes $P_{\text{i®MDNM¯}}$, i = 1, ..., n, and a set R of m types of resources $R_{\text{j®MDNM¯}}$, j = 1 ,..., m, where we assume the resources are serially reusable, Avail(j) units of resource j are available for allocation, and Claim(i,j) gives the maximum number of units of resource j which process i might possibly need at any point. The shortage of resource j relative to a subset of processes S®MDNM¯ is defined by

$$\text{Short}(S,j) = \sum_{i \text{ in } S} \text{Claim}(i,j) - \text{Avail}(j).$$

®
Short(S,j) represents the excess of demand for resource j from the processes in S over the total supply of j available in the system. If Short(S,j) < 0, there can never be a shortage of resource j, so the system can never be blocked on j. We are interested in conditions that guarantee the system is never blocked even when Short(S,j) > 0, for some resources j.

The resource requirements of the system are modeled by a shortage graph which summarizes the process-resource claims and potential resource shortages. The *shortage graph®MDNM¯* for a system (P, R, Claim, Avail) with respect to a subset of processes S in P®MDNM¯ is a labelled bipartite graph denoted by Shortage (S, P, R, Claim, Avail) (or G(S) for short) with parts $V_{\text{1®MDNM¯}}$ and $V_{\text{2®MDNM¯}}$ such that

(1) There is a vertex in $V_{\text{1®MDNM¯}}$(G) corresponding to each process in S, and a vertex in $V_{\text{2®MDNM¯}}$(G) for each resource class in R;

(2) There is an edge (p,r) between process vertex p and resource vertex r if and only if

(2a) Claim(p,r) > 0, that is, process p has a potential claim on an instance of resource r, and

(2b) Short(S,r) > 0, that is, there is a shortage of resource r relative to S;

(3) Each vertex has a label. Label(p) equals 1, for every process vertex p; Label(r) equals the potential shortfall of r: max{0, Short(S,r)}, for every resource vertex r.

Refer to Figure 8-4 for an example.

Figure 8-4 here

Let us define a *degree constrained matching®MDNM¯* M for a labelled graph G(V,E) as a set M of edges of G such that the number of edges of M incident at any vertex v of G is $\leq$ Label(v). Refer to Figures 8-5a and 8-6a for examples. A *maximum degree constrained matching®MDNM¯* is a degree constrained matching of maximum cardinality. If there exists a degree constrained matching for the shortage graph G(S) that matches every process vertex in G(S), then G(S) is said to have a *complete degree constrained matching®MDNM¯*.

4	{1,3}	1	3

Figure 8-2. Optimal Schedule for Figure 8-1.

®

The proof of the correctness of this scheduling algorithm is fairly difficult. We refer to Coffman and Denning (1973). However, it is easy to prove the following theorem.

THEOREM (SCHEDULING LOWER BOUND) Let a system of tasks constrained by a set of precedence constraints represented by a digraph G(V,E) be defined as above. Let $t_{\text{min®MDNM¯}}$ denote the minimum time in which the task system can be completed. Then,

$$t_{\text{min®MDNM¯}} \geq |V(G)| - |M(C(U(T(G)))|.$$

where M() returns a maximum matching on its argument graph, C() returns the complement of its argument, U() returns the undirected graph underlying its digraph argument, and T() returns the transitive closure of G.

The proof of the bound is simple. Observe that there is an edge between any pair of tasks in U(T(G)) which are either directly or indirectly dependent on one another. Therefore, C(U(T(G))) contains edges only between those tasks that are completely independent of one another. Consequently, if an optimal processing schedule schedules the pairs of tasks $T1_{\text{i®MDNM¯}}$, $T2_{\text{i®MDNM¯}}$, i = 1 ,..., k in parallel, the set of edges {($T1_{\text{i®MDNM¯}}$, $T2_{\text{i®MDNM¯}}$)} determines a matching in C(U(T(G))). Therefore, the maximum possible degree of parallelism attainable is constrained by the size of a maximum matching in C(U(T(G))). If a maximum matching has size m, at most m tasks can be parallelized. It follows that the optimal schedule must take time at least |V(G)| - m. This completes the proof.

Refer to Figure 8-3 for an example. We observe without proof that the optimal scheduling algorithm attains the lower bound. Observe also that not every maximum matching determines a feasible schedule. For example, consider the matching {($T_{\text{1®MDNM¯}}$,$T_{\text{4®MDNM¯}}$), ($T_{\text{2®MDNM¯}}$,$T_{\text{3®MDNM¯}}$)} for the graph in Figure 8-3b.

Figure 8-3 here

8-2-3 DEGREE CONSTRAINED MATCHING AND DEADLOCK FREEDOM

A deadlock avoidance algorithm allocates a set of shared resources among a set of processes in a way that avoids even the possibility of a deadlock. In contrast, a system of processes and resources is said to be *deadlock free®MDNM¯* if no deadlock is possible, regardless of how the resources are allocated. Since no special measures are needed to prevent deadlock, the overhead of a deadlock avoidance algorithm is avoided. We will describe a generalized matching property for a resource allocation graph that guarantees a system is deadlock free. But, first we will introduce a bigraph model of the resource requirements of a system of processors and processes.

8-2-2 MATCHING AND PROCESSOR SCHEDULING

There is a simple relationship between maximum matchings and optimal schedules for a system of tasks executing on a pair of parallel processors. Thus, suppose that each of a set of tasks takes unit time to execute and that the order in which the tasks can be scheduled is restricted by precedence constraints. The precedence constraints are represented by an acyclic digraph G(V,E), whose vertices correspond to the set of tasks, and whose edges represent the precedence constraints. We will show how to schedule these tasks in parallel on a pair of processors in such a way as to minimize the earliest completion time of the system. Then, we shall bound the completion time in terms of the size of a maximum matching on G.

The optimal scheduling algorithm is as follows. We first rank the vertices of G(V,E) according to a priority scheme defined as follows.

(1) Identify tasks $T_{\text{i1®MDNM¯}}$ to $T_{\text{ik®MDNM¯}}$ having no succcessors and assign them priorities 1 to k, respectively. Set Next to k + 1.

(2) Repeat (2a and b) until every task has been assigned a priority.

(2a) For each task T all of whose successor tasks $s_{\text{1®MDNM¯}}$, ..., $s_{\text{m®MDNM¯}}$ have been assigned a priority, let v(T) be an m-vector with components the priority labels of the tasks in ascending order.

(2b) Select the lexicographically smallest vector-labelled task T, and set priority(T) to Next. Then, set Next to Next + 1.

Figure 8-1 illustrates the procedure.

Figure 8-1 here

Once the priorities have been assigned, we schedule the processors according to the following rule.

Whenever a processor becomes free, assign X to execute the highest priority task remaining to be completed.

Figure 8-2 gives an optimal schedule for the example of Figure 8-1.

Time	Available	$P_{\text{1®MDNM¯}}$	$P_{\text{2®MDNM¯}}$
0	{9}	9	-
1	{5,8}	5	8
2	{4,7}	4	7
3	{1,2,6}	6	2

$|M_{i\ +\ 1\text{®MDNM¯}}| \ \geq$ min {max cardinality of an independent set in G'},

where the minimum is taken over all graphs G' of order |Mi| and size at most n.

Applied to our problem, Turan's theorem (1) implies that

$$|M_{i\ +\ 1\text{®MDNM¯}}| \ \ \geq \ \ |M_{i\text{®MDNM¯}}|^{\text{®MDSU¯2®MDNM¯}} \ / \ (2k + |M_{i\text{®MDNM¯}}|).$$

The lower bound for the value of i on termination follows readily. $|M_{0\text{®MDNM¯}}|$ must equal n, since $M_{0\text{®MDNM¯}}$ is just the original set of n values. The algorithm cannot stop until $|M_{i\text{®MDNM¯}}| = 1$. Therefore, the earliest time i at which the algorithm can terminate satisfies

$$i \ \geq \ \log(\log(n)) - c,$$

for some constant c, which completes the proof of the lower bound of the theorem. It remains to show the bound is sharp.

The preceding part of the proof demonstrates that the parallel maximum cannot be computed in faster than O(log(log(n))) time using n processors. A related result is obtainable if k < n processors are used. Using part (2) of Turan's Theorem, we can show the lower bound is attainable. For simplicity, we again assume k = n. We will design an optimal parallel algorithm for finding the maximum using a special class of extremal graphs to determine which elements to compare at each step. The extremal comparison graphs will be such that $|M_{i\text{®MDNM¯}}|$ is reduced to 1 in log(log(n)) steps.

The extremal graphs are just the graphs H(p,r) of Turan's Theorem. H(p,r) consists of r disjoint complete subgraphs. The comparisons for each complete subgraph yield exactly one potential maximum, or r candidates for maximum altogether. We will select a feasible H(p,r) with the correct number of vertices and edges that minimizes r. The feasible graphs are $H(|M_{i\text{®MDNM¯}}|,r)$ of order $|M_{i\text{®MDNM¯}}|$ and size at most k. By Turan's theorem, or directly by the definition of these graphs, the feasible graphs are those for which $(t - 1)(2|M_{i\text{®MDNM¯}}| - rt)/2 \leq k$, where t is the smallest integer as large as $|M_{i\text{®MDNM¯}}|/r$. We select that H(p,r) of order p and size at most k that minimizes r. If we use this choice of extremal graph to determine the comparisons at step i, the number of potential maxima at step i + 1 is bounded above by

$$|M_{i\ +\ 1\text{®MDNM¯}}| \leq \min \{ \ r \ 3 \ (t - 1)(2|M_{i\text{®MDNM¯}}| - rt)/2 \leq k \ \}.$$

which implies,

$$|M_{i\ +\ 1\text{®MDNM¯}}| \leq |M_{i\text{®MDNM¯}}|^{\text{®MDSU¯2®MDNM¯}} \ / \ (c\ k),$$

for some constant c. Since $|M_{0\text{®MDNM¯}}|$ equals n, $|M_{i\text{®MDNM¯}}|$ is reduced to 1 for some i less than log(log(n)) + c. This proves the sharpness of the lower bound.

Then,

(1) r is at least $p^{\text{®MDSU¯2®MDNM¯}}$ / (2k + p).

Furthermore, if we define the graphs H(p,r), for positive integers p and r (r < p), as the disjoint union of r - rt + p complete graphs K(t) and rt - p complete graphs K(t - 1), where t is the smallest integer as large as p/r, then,

(2) H(p,r) has size (t - 1)(2p - rt)/2, and has the minimum number of edges among all graphs of order p with vertex independence number at most r.

See Turan (1954) for a proof. The desired theoretical lower bound on the speed of parallel maximum selection is then given by the following theorem.

THEOREM (LOWER BOUND FOR PARALLEL MAXIMUM) Any algorithm based on binary comparisons and using n processors to find the maximum of n elements takes at least time O(log(log(n)). The bound is sharp.

The proof is as follows. Consider the state of affairs at the beginning of step i + 1 of the algorithm X. At this point, X can compare values from $M_{\text{i®MDNM¯}}$ either to each other or to values from M - $M_{\text{i®MDNM¯}}$, or compare values from M - $M_{\text{i®MDNM¯}}$ to each other. Comparisons between M - $M_{\text{i®MDNM¯}}$ values can yield no new information about the maximum, because every one of these values is already known by the end of step i to be excluded as a candidate for maximum. On the other hand, we cannot a priori exclude the possibility that in each comparison between a value from M - $M_{\text{i®MDNM¯}}$ and a value from $M_{\text{i®MDNM¯}}$, the value from $M_{\text{i®MDNM¯}}$ is greater; so these comparisons might also yield no additional information that could narrow the list of candidates for maximum. We will establish a lower bound under the assumption that this worst case occurs. Consequently, we only need to consider the effect that comparisons among values in $M_{\text{i®MDNM¯}}$ can have on the number of candidates for maximum.

Observe that at each step, the algorithm makes at most k (or n) comparisons. We model these comparisons by a graph G(V,E) where

(1) V(G) has |$M_{\text{i®MDNM¯}}$| vertices, one for each of the current candidates for maxima $M_{\text{i®MDNM¯}}$, andMDSD

(2) E(G) has at most k edges, one for each of the binary comparisons made at step i.

The size of $M_{\text{i+1®MDNM¯}}$, the updated list of candidate for maximum, is related to a feature of the model graph. Thus, suppose S is an independent set of vertices in G. S corresponds to a set of numbers no two of which are compared by the algorithm at step i. If it happens that each of the elements in S is greater than every value in $M_{\text{i®MDNM¯}}$ - S, any of the numbers (vertices) in S could be the maximum. It follows that |$M_{\text{i+1®MDNM¯}}$| could be (though not necessarily must be) as large as the largest set of independent vertices in any graph of order |$M_{\text{i®MDNM¯}}$| and size k (or n). Therefore, |$M_{\text{i + 1®MDNM¯}}$| must be at least as large as the smallest such independent set taken over all such graphs. That is, Therefore, the set ($V_{1®MDNM¯}$ - $W_{1®MDNM¯}$) U $W_{2®MDNM¯}$ is a vertex cover of cardinality |M|, since otherwise there is an edge ($w_{1®MDNM¯}$, y) with $w_{1®MDNM¯}$ in $W_{1®MDNM¯}$ and y not in $W_{2®MDNM¯}$, contrary to the conclusion that ADJ($W_{1®MDNM¯}$) = $W_{2®MDNM¯}$. This completes the proof of the theorem.

Regular bigraphs are 1-factorable by Koenig's theorem, although arbitrary bigraphs may not be. However, bigraphs can always be partitioned into a small number of edge disjoint matchings.

THEOREM (MATCHING PARTITION OF BIPARTITE GRAPHS) The edges of a bipartite graph G(V,E) can be partitioned into max(G) disjoint edge matchings.

The matchings guaranteed by this theorem are not necessarily spanning matchings, so this is not a factorization result. It follows from the theorem that if G is a bigraph, then ECHR(G) $\leq$ max(G).

8-2 MODELS

8-2-1 INDEPENDENCE NUMBER AND PARALLEL MAXIMUM

We have previously given a theoretical lower bound on the speed with which n mumbers could be sorted using a sequential binary comparison sort (Chapter 4). We will now establish a lower bound on the speed with which the largest of n numbers can be found using k parallel processors. The model for the problem is an undirected graph in which the result of a round of comparisons is represented by an independent sets of vertices. An optimal maximum value selection algorithm based on the lower bound analysis is also presented.

The model of parallel computation is CREW PRAM. That is, we assume the k processors can compare k not necessarily distinct pairs of values in constant time (see Chapter 9 for further discussion of shared memory models of parallel computation). The processors may read common memory locations concurrently, but make no concurrent writes to common locations. For simplicity, we assume the n numbers are distinct and the number k of processors equals n.

Let A denote an arbitrary binary comparison algorithm that uses k processors to find the largest value in a set M of n numbers. The algorithm consists of a sequence of constant time steps, where each of the k processors makes a comparison, in parallel. Let $M_{i®MDNM¯}$ denote the subset of values in M that have not yet been shown to be smaller than any of the other values in M by the end of step i. In other words, each value in $M_{i®MDNM¯}$ remains a candidate for the maximum value at the next step i + 1 of the algorithm. However, every value in M - $M_{i®MDNM¯}$ has definitely been excluded from being the maximum by step i + 1. Naturally, the algorithm terminates when |$M_{i®MDNM¯}$| = 1.

To prove the lower bound on the termination time, we require the following extremal result.

THEOREM (MINIMAX INDEPENDENT SET OF TURAN) Let G(V,E) be a graph of order p and size k, and let r be the order of the largest vertex independent set in G.

FACTORS

A *factor®MDNM¯* of a graph G(V,E) is a spanning subgraph of G. If G can be expressed as the edge sum of factors of G, the edge sum is ®called a *factorization®MDNM¯* of G. An *r-factor®MDNM¯* of a graph G(V,E) is an r-regular factor of G. Thus, a 1-factor of G is a perfect matching of G, while a 2-factor of G is either a spanning cycle of G, or a disjoint union of cycles which together span G. G is said to be *r-factorable®MDNM¯* if it has a factorization consisting only of r-factors. Koenig's Theorem (Section 1-4) shows regular bigraphs are 1-factorable. The following theorems are famous.

THEOREM (TUTTE'S 1-FACTOR CONDITION) A nontrivial graph G(V,E) has a 1-factor if and only if for every proper subset S of V(G), the number of components of G - S of odd order is at most |S|.

THEOREM (PETERSEN'S 2-FACTOR CONDITION) A nonempty graph G(V,E) is 2-factorable if and only if G is 2r-regular for some r $\geq$ 1.

THEOREM (PETERSEN'S CHARACTERIZATION OF CUBIC GRAPHS) Every bridgeless cubic graph can be expressed as the edge sum of a 1-factor and a 2-factor.

The Petersen graph (see Section 1-4) shows not every bridgeless cubic graph is 1-factorable.

There are noteworthy special results on covers and matchings for bigraphs. We have already established (Section 1-3) a noncontracting condition for the existence of a spanning matching on a bigraph. The following theorem strengthens inequalities we have given for general graphs.

THEOREM (BIPARTITE MATCHING, COVERING, AND INDEPENDENCE) If G(V,E) is a bipartite graph with no isolated vertices, then

(1) Max-Match(G) = Min-VCov(G), and

(2) Max-Indep(G) = Min-ECov(G).

We shall only prove statement (1). By the inequalities for the general theorem, we already know that Max-Match(G) $\leq$ Min-VCov(G), so we only need to prove that Max-Match(G) $\geq$ Min-VCov(G). Denote the bipartite parts of G by V$_{\text{1®MDNM}}$¯ and V$_{\text{2®MDNM}}$¯, and let M be a maximum matching of cardinality Max-Match(G). If G has a perfect matching, then Max-Match(G) = Min-VCov(G), and we are done.

Otherwise, let X be the set of vertices in G not covered by M. Thus, |M| = |V$_{\text{1®MDNM}}$¯| - |X|. Let R be the set of vertices in G reachable from X by alternating paths with respect to M. Let W$_{\text{i®MDNM}}$¯ (i = 1,2) be the part of R in V$_{\text{i®MDNM}}$¯. By the definition of X, W$_{\text{1®MDNM}}$¯ - X is matched by M to W$_{\text{2®MDNM}}$¯, so |W$_{\text{2®MDNM}}$¯| = |W$_{\text{1®MDNM}}$¯| - |X|, and W$_{\text{2®MDNM}}$¯ is contained in ADJ(W$_{\text{1®MDNM}}$¯). Conversely, W$_{\text{1®MDNM}}$¯ is contained in ADJ(W$_{\text{2®MDNM}}$¯), since G contains an alternating path from X to w, for every vertex w in ADJ(W$_{\text{1®MDNM}}$¯). Therefore, ADJ(W$_{\text{1®MDNM}}$¯) = W$_{\text{2®MDNM}}$¯, and |ADJ(W$_{\text{1®MDNM}}$¯)| = |W$_{\text{2®MDNM}}$¯| = |W$_{\text{1®MDNM}}$¯| - |X|.

a vertex v in V - S which is not adjacent to some vertex of S, then S U {v} is an independent set of cardinality Max-Indep(G) + 1. Therefore, S must be a vertex dominating set, whence it follows that Min-VDom(G) $\leq$ Max-Indep(G).

It follows from this theorem that if the size of a matching equals the size of a vertex cover, the matching is a maximum matching and the vertex cover is a minimum vertex cover. Similar remarks hold for the other quantities.

THEOREM (GALLAI'S COMPLEMENTARITY RELATIONS) If G(V,E) is a graph containing no isolated vertices, then

(1) Max-Match(G) + Min-ECov(G) = |V(G)|, and

(2) Max-Indep(G) + Min-VCov(G) = |V(G)|.

To prove (1), we will first show that Max-Match(G) + Min-ECov(G) $\leq$ |V(G)|. Let $E_{\text{ind®MDNM}}$¯ be an independent set of edges of G of cardinality Max-Match(G). Thus, $E_{\text{ind®MDNM}}$¯ covers 2 Max-Match(G) vertices of G. For each vertex in G not covered by an edge of $E_{\text{ind®MDNM}}$¯, select an additional edge from E(G) and denote the union of these edges by $E_{\text{extra®MDNM}}$¯. Then, $E_{\text{ind®MDNM}}$¯ U $E_{\text{extra®MDNM}}$¯ is an edge cover of G, so that 3$E_{\text{ind®MDNM}}$¯ U $E_{\text{extra®MDNM}}$¯| is at least Min-ECov(G). But, |$E_{\text{ind®MDNM}}$¯| + |$E_{\text{ind®MDNM}}$¯ U $E_{\text{extra®MDNM}}$¯| = |V|; so Max-Match(G) + Min-ECov(G) $\leq$ |V(G)|. To prove that Max-Match(G) + Min-ECov(G) $\geq$ |V(G)|, let $E_{\text{min®MDNM}}$¯ be a minimum edge cover of G. Then, the subgraph determined by $E_{\text{min®MDNM}}$¯ is acyclic. If we select an edge from each of its components, and denote the resulting set of edges by $E_{\text{comp®MDNM}}$¯, then |$E_{\text{comp®MDNM}}$¯| $\leq$ Max-Match(G), and |$E_{\text{min®MDNM}}$¯| + |$E_{\text{comp®MDNM}}$¯| = |V|. It follows that Max-Match(G) + Min-ECov(G) $\geq$ |V(G)|, which completes the proof of (1).

To prove (2), first observe that S is an independent set of vertices of G(V,E) if and only if V - S is a vertex cover of G. Then, let $S_{\text{max®MDNM}}$¯ be a maximum independent set, and let $T_{\text{min®MDNM}}$¯ be a minimum vertex cover of G. By observation, S' = V - $S_{\text{max®MDNM}}$¯ is a vertex cover, and T' = T - $T_{\text{min®MDNM}}$¯ is an independent set. Therefore,

|S'| = |V - $S_{\text{max®MDNM}}$¯| = |V| - Max-Indep(G) $\geq$ Min-VCov(G),

while,

|T'| = |V - $T_{\text{min®MDNM}}$¯| = |V| - Min-VCov(G) $\leq$ Max-Indep(G).

Combining the inequalities, we obtain (2).

8 COVERS, DOMINATION, INDEPENDENT SETS, MATCHINGS, AND FACTORS

8-1 BASIC CONCEPTS

We now introduce the elements of the theory of graph covering and related invariants. If G(V,E) is a graph, then a *vertex®MDNM¯* of G is said to cover any edge of G it is incident with®MDNM¯, and similarly, an *edge®MDNM¯* of G is said to *cover its endpoints®MDNM¯*. A set of vertices in G that cover every edge of G is called a *vertex cover®MDNM¯*; while a set of edges in G that cover every vertex of G is called an *edge cover®MDNM¯*.

Domination is a subtly different notion. While a *vertex®MDNM¯* is said to cover an incident edge, it is said to *dominate®MDNM¯* both itself and any adjacent vertex. A *vertex dominating set®MDNM¯* is a set of vertices S such that every vertex in G is dominated by some vertex in S. Similarly, an *edge®MDNM¯* is said to *dominate®MDNM¯* the edges incident to its endpoints, as well as itself. An *edge dominating set®MDNM¯* is a set of edges X such that every edge in G is dominated by some edge in X.

The notion of an *independent set®MDNM¯* is complementary to that of a cover. While a cover requires adjacency, an independent set requires nonadjacency. We will call a set of vertices no two of which are adjacent an *independent set of vertices®MDNM¯*. We will call a set of edges no two of which are adjacent an *independent set of edges®MDNM¯* or a *matching®MDNM¯*. An independent set of vertices is said to be *maximal®MDNM¯* if any vertex not in the set is dominated by a vertex in the set.

There are simple relations between the extreme values of the sizes of covers, dominating sets, and independent sets.

Minimum cardinality of a vertex cover of G:	Min-VCov (G)
Minimum cardinality of an edge cover of G:	Min-ECov (G)
Minimum cardinality of a vertex dominating set:	Min-VDom (G)
Minimum cardinality of an edge dominating set:	Min-EDom (G)
Maximum cardinality of a matching on G:	Max-Match(G)
Maximum cardinality of a vertex independent set of G:	Max-Indep(G)

Each of these invariants has a simple physical interpretation. For example, suppose we can install "service centers" at a subset of the vertices of a graph, where each center can provide service both for its home vertex and adjacent vertices. Then, the minimum number of such centers needed to provide service to the whole graph equals Min-VDom(G). The following theorems are easily proved.

THEOREM (COVERS, DOMINATING SETS, MATCHINGS, AND INDEPENDENT SETS) If G(V,E) is a graph with no isolated vertices, then

(1) Min-EDom(G) $\leq$ Max-Match(G) $\leq$ Min-VCov(G), and

(2) Min-VDom(G) $\leq$ Max-Indep(G) $\leq$ Min-ECov(G).

As an example, consider the proof of Min-VDom(G) $\leq$ Max-Indep(G). Let S be an independent set of vertices of G such that |S| = Max-Indep(G). If there exists