Chess Problem Solver Project
1. Abstract

The goal of this paper is to demonstrate a simple implementation of a parallel chess problem solving program. A solution to a chess problem means that one or more set of moves exist such that a player can force checkmate on the opponent, regardless of the moves the opponent makes during their turn. The program as designed should efficiently accommodate a fairly large number of processors estimated in the range of a few to several hundred assuming a reasonably large problem size. It has also been designed to accommodate future enhancements such as dynamic load balancing and a hierarchical group-based approach. Efficiency factors of the basic system and the enhanced system will be discussed, along with a demonstration of the results of several problems.

2. Problem Characteristics

The chess solver problem is an example of a very large computation with an extremely small amount of inter-process data dependencies. In other words, the communication to computation ratio is extremely low for even smaller problems with few moves to search. The simplest approach to solving this problem is by using a search tree. The problem is successively broken down using all possible moves for any given position until all possible results of each branch is determined. At that time, each of the branches can be examined to determine the overall result of the root node. The result determination method will depend on whose turn it is at the root node. If it’s the solving players turn, then any branch representing a possible solution is a valid solution for that root node. No solutions at any of the branches means that the root node does not represent a possible solution. If it’s not the solving players turn, then all branches underneath the root node must result in possible solutions in order for the root to represent a valid solution. The reason behind the different algorithms for the players is because it is assumed that the solving player will always make the proper move in order to achieve checkmate and the opponent will always make the proper move in order to avoid checkmate.

To implement this system, a message passing master/slave process approach using PVM was chosen. Also, distributed task queues were chosen to keep communication to a minimum and to enable low overhead dynamic load balancing. Due to the size of the computations the tasks will be coarse grain, again to keep communication to a minimum. The system is also designed for future flexibility. The specifics will be discussed in detail in the following sections.

3. Basic Software Requirements

Three basic items were required to implement the system: a basic data unit to describe a problem, a basic software function to break down a problem, and a tree search routine to find the result of a problem.

The basic data unit used will be referred to as a “problem” or a “task” interchangeably. It contains all the necessary information about the state of a given chess position. These include whose turn it is to move, what player the problem is being solved for, the maximum number of moves to search, the move that was made to get to the position, and the status of the pieces on the board. The status of the pieces on the board refers to not only the pieces and their locations, but also whether or not they have moved for castling purposes and whether or not the piece is able to be captured en-passent.

The basic software function was created to break down a given problem. It takes a given problem and breaks it into a list of new problems using all possible moves. Each new problem will simply be a copy of the original with only a few changes: it reflects the branching move and the maximum search level remaining counter is decremented by one. As will be discussed, this function is used to create the levels and branches in the search tree.
The basic tree search routine was created to examine each problem node in the tree and determine the necessary action. There are only two possibilities: the problem represents an end node, or further investigation of the problem is necessary. To be an end node, the problem must represent either a possible or an impossible solution. A possible solution means that the solving player has checkmated the opponent. An impossible solution means that either a stalemate has occurred, the opponent has checkmated the solving player, or further investigation is still necessary but the remaining search levels has reached zero.

The following figure illustrates the tree branching data structure used in software. Each branch level in the tree is simply stored as a singly linked list. An array of pointers keeps track of each of the branch levels with a start-of-list pointer and a pointer to the current node being examined. The root node of any given branch is the current task being examined at the next higher tree level. When a search is complete on the lowest tree level, the obtained result (possible solution, impossible solution) is passed up to its root node. The lowest tree level is then destroyed, the current task pointer is advanced at the next higher level, and if further investigation of the node is required, then another level of the tree will be created representing all of the branches of that root.

### Search Tree Software Implementation

![Diagram of Search Tree](image)

4. **PVM Implementation**

A standard master/slave program using PVM was used to take advantage of the high degree of parallelism inherent in this type of problem. It’s important to note that since spawning slave processes is very time consuming, this design reuses slave processes during the course of the computation. Only when the program is complete does the master kill the slave processes. The only cost tradeoff with this method is a
single value added in the message from the master to the slave that indicates the command type. A very profitable tradeoff.

The master process is responsible for getting the initial problem from a text format and converting it to an original problem task. The text file includes the desired number of processes, the maximum number of moves to search, the player who has the initial move (who is also the solving player), and the initial status of pieces on the board. All necessary slave processes are created first. The master must then break the original problem into a list of tasks using the “break down problem” function. The tasks will then be handed out to idle slave processes, one task per process, until all tasks have been completed and the results returned by the slaves. This means that when a process has completed a task and returned the result, the master will then send the slave another task if available. The master maintains a collective list of results compiled from all the slaves during the course of the program. When the computation is complete, it simply displays the list of results to the user.

The slave process is simply responsible for receiving command messages from the master and executing them. The master-to-slave commands defined at this stage of the development include: solve problem, transfer work, and terminate.

The solve problem message is followed by the actual problem data to be solved. The slave will solve the problem and is expected to return the results to the master.

Although the transfer work command has not yet been physically implemented, it is designed to allow for dynamic load balancing. Near the end of the computation, the master will run out of tasks to give slave processes as they complete. This message will allow the master to designate one slave that is busy to give another some of its work. The message data will be followed by the idle process id to transfer the work to.

The terminate message requires no extra data in the message and will simply instruct the slave process to terminate itself. In the flat master-slave approach, this message really provides an insignificant improvement over having the master simply kill each of the slaves itself, but will become important in a hierarchical based group design as discussed in the next section.

The following figure illustrates the master-slave approach as used in this project.

![System Layout](image-url)
5. Future Enhancement by Grouping

The current design as presented will work very well with just about any real system. However, for a system that has a huge number of processors, modifications may be required. A simple, yet extremely effective enhancement that can be made to the system is to use a hierarchical approach using groups of processors. In this enhanced system, the slave in the previous system model will be replaced by a group of processors. Each group will internally have a head process determined, for example, by the lowest tid of the group. The master process will communicate tasks to only the head processes of each group and those head processes will further break down the task it is given and will send items from its task list to the slaves in its group.

To illustrate the effectiveness of this approach, suppose that a given problem set requires usage of 50,000 processors to compute the results of a given problem set in a reasonable time frame. Using the existing approach, the master process would have to keep track of and communicate with 50,000 slave processes. Even with very coarse grain tasks and a reasonably short network latency, this approach will lead to many processes sitting idle while either waiting for instructions or waiting to report results. If the system was instead broken into 200 groups of 250 processors each, the master will now only be required to communicate with 200 processes, and the group-master processes will each only have to maintain communications with 250 other processes. Although this example illustrates an extremely unlikely number of processors, it is effective in showing how well the group approach can be structured for handling large systems of nearly any size.

6. Performance Issues

The system is limited in efficiency by the number of processes that the master must communicate with and the granularity of the tasks. With fixed coarse grain tasks, the system will run very efficiently with a smaller number of processors, but the efficiency will decrease as more processors are added. The reason behind this is that when a slave task needs to communicate with the master, the master might possibly be busy communicating with another task. The slave will then have to wait until the master completes its transaction(s) before performing its own communications and getting back to work. Obviously when there are fewer processors in a system with a given coarse task granularity, probability dictates that there will be few stalled slaves and the system will consequently run efficiently. As the number of processors in the system increases, the probability of stalled slaves increases which will lead to less efficiency.

In a system with a very large number of processors, it is obvious that the hierarchical group approach will help to greatly decrease the number of processors that the master or any group-master will need to communicate with. This will greatly decrease the probability of stalled slave processes and will consequently increase the system efficiency.

7. Results

Several problems have been run to test the effectiveness of the parallel program. Six output scripts are shown. Note that the original problem is shown twice in each example, at the beginning before the problem has been solved and at the end just before the solution variations are displayed.

Of the examples displayed, the first five files (input1.txt – input5.txt) are just simple examples to demonstrate basic problem solving ability. The sixth file (input6.txt) is actually quite a challenging puzzle, even for an intermediate player. It was taken from a chess puzzle book titled “200 Classic Chess Puzzles” written by Martin Greif.
Note that the solution results sometimes contain various numbers of moves. This is due to the fact that all possible opponent move variations are displayed. Keep in mind that the solving players moves will force the opponent to choose one of the displayed solutions.

```
[File “input1.txt”]
8
4
1
WRa6 0 0
WKg6 0 0
BKh8 0 0
BPa7 0 0

Script started on Thu May 20 12:57:39 1999
[crb4906@mealy]$ ~/pvm3/bin/LINUX/chess < input1.txt

remaining srch levels = 4
White's turn flag = 1
solve for white flag = 1
num_processes = 8
- - - - - - k 0 0 0 0 0 0 0 0 0 0 0 0 1 0
  0 0 0 0 0 0 0 0 0 0 0 0 1 1 1
p - - - - - - - 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1
  0 0 0 0 0 0 1 1 1 0 1 0 0 0 0 0
R - - - - - K - 0 1 1 1 1 2 1 1 0 1 0 0 0 0 0
  1 0 0 0 0 1 1 1 0 0 0 0 0 0 0
- - - - - - - - 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
- - - - - - - - 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
- - - - - - - - 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
- - - - - - - - 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
- - - - - - - - 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
Data sent to process 262147
Data sent to process 262148
Data sent to process 262149
Data sent to process 262150
Data sent to process 262151
Data sent to process 262152
Data sent to process 262153
Data sent to process 262154
sent initial tasks
```
received response from 262147, 7 tasks still running.
a6 to a7, h8 to g8, a7 to a8,
Data sent to process 262147
received response from 262148, 7 tasks still running.
No solution found. Data sent to process 262148
received response from 262150, 7 tasks still running.
No solution found. Data sent to process 262150
received response from 262151, 7 tasks still running.
No solution found. Data sent to process 262151
received response from 262152, 7 tasks still running.
No solution found. Data sent to process 262152
received response from 262154, 7 tasks still running.
a6 to c6, a7 to a6, c6 to c8,
a6 to c6, a7 to a5, c6 to c8,
a6 to c6, h8 to g8, c6 to c8,
Data sent to process 262154
received response from 262147, 7 tasks still running.
a6 to d6, a7 to a6, d6 to d8,
a6 to d6, a7 to a5, d6 to d8,
a6 to d6, h8 to g8, d6 to d8,
Data sent to process 262147
received response from 262149, 7 tasks still running.
No solution found. Data sent to process 262149
received response from 262153, 7 tasks still running.
No solution found. Data sent to process 262153
received response from 262153, 7 tasks still running.
No solution found. received response from 262150, 6 tasks still running.
No solution found. received response from 262151, 5 tasks still running.
No solution found. received response from 262152, 4 tasks still running.
No solution found. received response from 262147, 3 tasks still running.
No solution found. received response from 262148, 2 tasks still running.
a6 to e6, a7 to a6, e6 to e8,
a6 to e6, a7 to a5, e6 to e8,
a6 to e6, h8 to g8, e6 to e8,
received response from 262154, 1 tasks still running.
No solution found. received response from 262149, 0 tasks still running.

No solution found. Terminating process 262147
Terminating process 262148
Terminating process 262149
Terminating process 262150
Terminating process 262151
Terminating process 262152
Terminating process 262153
Terminating process 262154

--- k 0 0 0 0 0 0 0 0 0 0 0 0 1 0
p 1 0 0 0 1 1 1 0 0 0 0 0 1 1
R 0 1 1 1 1 2 1 1 0 1 0 0 0 0 0
--- 1 0 0 0 1 1 1 0 0 0 0 0 0 0
--- 1 0 0 0 0 0 0 0 0 0 0 0 0 0
--- 1 0 0 0 0 0 0 0 0 0 0 0 0 0
--- 1 0 0 0 0 0 0 0 0 0 0 0 0 0
--- 1 0 0 0 0 0 0 0 0 0 0 0 0 0
--- 1 0 0 0 0 0 0 0 0 0 0 0 0 0

Solution List:

a6 to a7, h8 to g8, a7 to a8,
a6 to c6, a7 to a6, c6 to c8,
a6 to c6, a7 to a5, c6 to c8,
a6 to c6, h8 to g8, c6 to c8,
a6 to d6, a7 to a6, d6 to d8,
a6 to d6, a7 to a5, d6 to d8,
a6 to d6, h8 to g8, d6 to d8,
a6 to e6, a7 to a6, e6 to e8,
a6 to e6, a7 to a5, e6 to e8,
a6 to e6, h8 to g8, e6 to e8,
[crb4906@mealy]$ exit

exit

Script done on Thu May 20 12:57:56 1999
Script started on Thu May 20 12:58:07 1999

remaining srch levels = 4

White's turn flag = 1

solve for white flag = 1

num_processes = 8

Data sent to process 262156
Data sent to process 262157
Data sent to process 262158
Data sent to process 262159
Data sent to process 262160
Data sent to process 262161
Data sent to process 262162
Data sent to process 262163
sent initial tasks
received response from 262156, 7 tasks still running.
a6 to a7, b7 to b6, a7 to a8,
a6 to a7, b7 to b5, a7 to a8,
a6 to a7, c7 to c6, a7 to a8,
a6 to a7, c7 to c5, a7 to a8,
a6 to a7, d7 to d6, a7 to a8,
a6 to a7, d7 to d5, a7 to a8,
a6 to a7, e7 to e6, a7 to a8,
a6 to a7, e7 to e5, a7 to a8,
a6 to a7, h8 to g8, a7 to a8,
Data sent to process 262156
received response from 262157, 7 tasks still running.
No solution found. Data sent to process 262157
received response from 262162, 7 tasks still running.
No solution found. Data sent to process 262162
received response from 262163, 7 tasks still running.
No solution found. Data sent to process 262163
received response from 262158, 7 tasks still running.
No solution found. Data sent to process 262158
received response from 262161, 7 tasks still running.
No solution found. Data sent to process 262161
received response from 262160, 7 tasks still running.
No solution found. Data sent to process 262160
received response from 262159, 7 tasks still running.
No solution found. Data sent to process 262159
received response from 262156, 7 tasks still running.
No solution found. received response from 262157, 6 tasks still running.
No solution found. received response from 262163, 5 tasks still running.
No solution found. received response from 262158, 4 tasks still running.
No solution found. received response from 262162, 3 tasks still running.
No solution found. received response from 262161, 2 tasks still running.
No solution found. received response from 262159, 1 tasks still running.
No solution found. received response from 262160, 0 tasks still running.
No solution found. Terminating process 262156
Terminating process 262157
Terminating process 262158
Terminating process 262159
Terminating process 262160
Terminating process 262161
Terminating process 262162
Terminating process 262163

- - - - - - k  0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
p p p p p - - 1 0 0 0 0 1 1 1 0 0 0 0 0 1 1
R - - - - - K  0 1 1 1 1 2 1 1 1 2 2 2 1 1 0 0
- - - - - - - 1 0 0 0 0 1 1 1 0 0 0 0 0 0 0 0
- - - - - - - 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
- - - - - - - 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
- - - - - - - 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
- - - - - - - 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Solution List:

a6 to a7, b7 to b6, a7 to a8,
a6 to a7, b7 to b5, a7 to a8,
a6 to a7, c7 to c6, a7 to a8,
a6 to a7, c7 to c5, a7 to a8,
a6 to a7, d7 to d6, a7 to a8,
a6 to a7, d7 to d5, a7 to a8,
a6 to a7, e7 to e6, a7 to a8,
a6 to a7, e7 to e5, a7 to a8,
a6 to a7, h8 to g8, a7 to a8,

[crb4906@mealy]$ exit

exit

Script done on Thu May 20 12:58:50 1999

[File “input3.txt”]
8
4
1
Script started on Thu May 20 12:55:24 1999

~/pvm3/bin[crb4906@amdahl]$ ~/pvm3/bin/LINUX/chess < input3.txt

remaining srch levels = 4

White's turn flag = 1
solve for white flag = 1
num_processes = 8

Data sent to process 262147
Data sent to process 262148
Data sent to process 262149
Data sent to process 262150
Data sent to process 262151
Data sent to process 262152
Data sent to process 262153
Data sent to process 262154

sent initial tasks
received response from 262151, 7 tasks still running.

No solution found. Data sent to process 262151
received response from 262147, 7 tasks still running.

No solution found. Data sent to process 262147
received response from 262153, 7 tasks still running.
No solution found. Data sent to process 262153
received response from 262152, 7 tasks still running.
No solution found. Data sent to process 262152
received response from 262148, 7 tasks still running.
No solution found. Data sent to process 262148
received response from 262154, 7 tasks still running.
No solution found. Data sent to process 262154
received response from 262150, 7 tasks still running.
No solution found. received response from 262149, 6 tasks still running.
No solution found. received response from 262151, 5 tasks still running.
No solution found. received response from 262147, 4 tasks still running.
No solution found. received response from 262153, 3 tasks still running.
No solution found. received response from 262152, 2 tasks still running.
No solution found. received response from 262154, 1 tasks still running.
No solution found. received response from 262148, 0 tasks still running.
No solution found. Terminating process 262147
Terminating process 262148
Terminating process 262149
Terminating process 262150
Terminating process 262151
Terminating process 262152
Terminating process 262153
Terminating process 262154
- - - - - - - k  0 0 0 0 0 0 0 0 0 0 0 0 0 2 1
p p p p p - - -  1 0 0 0 0 1 1 1 0 0 0 0 1 2 1
R - - - - - K -  0 1 1 1 1 2 1 1 1 2 2 2 2 0 0
- - - - - -  1 0 0 0 1 1 1 0 0 1 1 0 0 0
- - - - - -  1 0 0 0 0 0 0 0 1 1 2 0 0 0
- - - - - -  1 0 0 0 0 0 0 0 1 1 0 1 0 0 0
b - n - - - -  1 0 0 0 0 0 0 0 1 0 0 0 0 0
b - - - - - -  0 0 0 0 0 0 0 0 1 1 0 1 0 0 0

Solution List:
No solution found.

[crb4906@amdahl]$ exit

exit

Script done on Thu May 20 12:59:50 1999

[File "input4.txt"]
8
4
1
WRa6 1 0
WKg6 1 0
WBa1 1 0
BKh8 1 0
BPg7 1 0

Script started on Thu May 20 12:59:01 1999

[crb4906@mealy]$ ~/pvm3/bin/LINUX/chess < input4.txt

remaining srch levels = 4

White's turn flag = 1
solve for white flag = 1
num_processes = 8

- - - - - - k 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
- - - - - - p - 1 0 0 0 0 1 2 1 0 0 0 0 0 0 1 1
R - - - - - K - 0 1 1 1 1 3 1 1 0 0 0 0 0 1 0 1
- - - - - - - - 1 0 0 0 1 1 1 1 0 0 0 0 0 0 0 0
- - - - - - - - 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
- - - - - - - - 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
- - - - - - - - 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
B - - - - - - - - 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Data sent to process 262165
Data sent to process 262166
Data sent to process 262167
Data sent to process 262168
Data sent to process 262169
Data sent to process 262170
Data sent to process 262171
Data sent to process 262172
sent initial tasks
received response from 262165, 7 tasks still running.
a1 to b2, h8 to g8, a6 to a8,
Data sent to process 262165
received response from 262166, 7 tasks still running.
a1 to c3, h8 to g8, a6 to a8,
Data sent to process 262166
received response from 262167, 7 tasks still running.
a1 to d4, h8 to g8, a6 to a8,
Data sent to process 262167
received response from 262168, 7 tasks still running.
a1 to e5, h8 to g8, a6 to a8,
Data sent to process 262168
received response from 262169, 7 tasks still running.
a1 to f6, g7 to f6, a6 to a8,
a1 to f6, h8 to g8, a6 to a8,
Data sent to process 262169
received response from 262170, 7 tasks still running.
a1 to g7, h8 to g8, a6 to a8,
Data sent to process 262170
received response from 262171, 7 tasks still running.
a6 to a7, h8 to g8, a7 to a8,
Data sent to process 262171
received response from 262172, 7 tasks still running.
a6 to a8,
Data sent to process 262172
received response from 262165, 7 tasks still running.
a6 to a5, h8 to g8, a5 to a8,
Data sent to process 262165
received response from 262166, 7 tasks still running.
a6 to a4, h8 to g8, a4 to a8,
Data sent to process 262166
received response from 262166, 7 tasks still running.
a6 to a3, h8 to g8, a3 to a8,
Data sent to process 262167
received response from 262168, 7 tasks still running.
a6 to a2, h8 to g8, a2 to a8,
Data sent to process 262168
received response from 262169, 7 tasks still running.
a6 to b6, h8 to g8, b6 to b8,
Data sent to process 262169
received response from 262170, 7 tasks still running.
a6 to c6, h8 to g8, c6 to c8,
received response from 262171, 6 tasks still running.
a6 to d6, h8 to g8, d6 to d8,
received response from 262172, 5 tasks still running.
a6 to e6, h8 to g8, e6 to e8,
received response from 262165, 4 tasks still running.
No solution found. received response from 262166, 3 tasks still running.
No solution found. received response from 262167, 2 tasks still running.
No solution found. received response from 262169, 1 tasks still running.
No solution found. received response from 262168, 0 tasks still running.
No solution found. Terminating process 262165
Terminating process 262166
Terminating process 262167
Terminating process 262168
Terminating process 262169
Terminating process 262170
Terminating process 262171
Terminating process 262172

- - - - - - k   1 0 0 0 0 0 0 0 0 0 0 1 0
- - - - - - p -   1 0 0 0 0 1 2 1 0 0 0 0 0 1 1
R - - - - - K -   0 1 1 1 1 3 1 1 0 0 0 0 1 0 1
- - - - - - - -   1 0 0 0 1 1 1 1 0 0 0 0 0 0 0
- - - - - - - -   1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
Solution List:

a1 to b2, h8 to g8, a6 to a8,
a1 to c3, h8 to g8, a6 to a8,
a1 to d4, h8 to g8, a6 to a8,
a1 to e5, h8 to g8, a6 to a8,
a1 to f6, g7 to f6, a6 to a8,
a1 to f6, h8 to g8, a6 to a8,
a1 to g7, h8 to g8, a6 to a8,
a6 to a7, h8 to g8, a7 to a8,
a6 to a8,
a6 to a5, h8 to g8, a5 to a8,
a6 to a4, h8 to g8, a4 to a8,
a6 to a3, h8 to g8, a3 to a8,
a6 to a2, h8 to g8, a2 to a8,
a6 to b6, h8 to g8, b6 to b8,
a6 to c6, h8 to g8, c6 to c8,
a6 to d6, h8 to g8, d6 to d8,
a6 to e6, h8 to g8, e6 to e8,

[crb4906@mealy]$ exit
exit
Script done on Thu May 20 12:59:17 1999

[File "input5.txt"]
8
5
1
WRa6 1 0
WKg6 1 0
WBal 1 0
BKh8 1 0
BPg7 1 0

Script started on Thu May 20 12:59:24 1999
[crb4906@mealy]$ ~/pvm3/bin/LINUX/chess < input5.txt

remaining srch levels = 5

White's turn flag = 1
solve for white flag = 1
num_processes = 8

- - - - - - k 1 0 0 0 0 0 0 0 0 0 0 0 1 0
- - - - - - p - 1 0 0 0 0 1 2 1 0 0 0 0 0 1 1
R - - - - - K - 0 1 1 1 1 3 1 1 0 0 0 0 1 0 1
- - - - - - - - 1 0 0 1 1 1 1 0 0 0 0 0 0 0 0
- - - - - - - - 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
- - - - - - - - 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0
- - - - - - - - 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
B - - - - - - - 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Data sent to process 262174
Data sent to process 262175
Data sent to process 262176
Data sent to process 262177
Data sent to process 262178
Data sent to process 262179
Data sent to process 262180
Data sent to process 262181

sent initial tasks
received response from 262181, 7 tasks still running.
a6 to a8,

Data sent to process 262181

received response from 262179, 7 tasks still running.
a1 to g7, h8 to g8, a6 to a8,
a1 to g7, h8 to g8, g7 to h6, g8 to h8, a6 to a8,
a1 to g7, h8 to g8, g7 to f6, g8 to f8, a6 to a8,
Data sent to process 262179
received response from 262180, 7 tasks still running.
a6 to a7, h8 to g8, a1 to f6, g7 to f6, a7 to a8,
a6 to a7, h8 to g8, a1 to f6, g8 to h8, a7 to a8,
a6 to a7, h8 to g8, a1 to f6, g8 to f8, a7 to a8,
a6 to a7, h8 to g8, a1 to f6, g8 to f8, a7 to a8,
a6 to a7, h8 to g8, a7 to a8,
a6 to a7, h8 to g8, a7 to f7, g8 to h8, f7 to f8,

Data sent to process 262180
received response from 262174, 7 tasks still running.
a1 to b2, h8 to g8, a6 to a8,
a1 to b2, h8 to g8, b2 to a3, g8 to h8, a6 to a8,
a1 to b2, h8 to g8, b2 to f6, g7 to f6, a6 to a8,
a1 to b2, h8 to g8, b2 to f6, g8 to h8, a6 to a8,
a1 to b2, h8 to g8, b2 to f6, g8 to f8, a6 to a8,

Data sent to process 262174
received response from 262181, 7 tasks still running.
a6 to a5, h8 to g8, a1 to f6, g7 to f6, a5 to a8,
a6 to a5, h8 to g8, a1 to f6, g8 to h8, a5 to a8,
a6 to a5, h8 to g8, a1 to f6, g8 to f8, a5 to a8,
a6 to a5, h8 to g8, a5 to a8,
a6 to a5, h8 to g8, a5 to f5, g8 to h8, f5 to f8,

Data sent to process 262181
received response from 262175, 7 tasks still running.
a1 to c3, h8 to g8, a6 to a8,
a1 to c3, h8 to g8, c3 to b4, g8 to h8, a6 to a8,
a1 to c3, h8 to g8, c3 to f6, g7 to f6, a6 to a8,
a1 to c3, h8 to g8, c3 to f6, g8 to h8, a6 to a8,
a1 to c3, h8 to g8, c3 to f6, g8 to f8, a6 to a8,

Data sent to process 262175
received response from 262177, 7 tasks still running.
a1 to e5, h8 to g8, a6 to a8,
a1 to e5, h8 to g8, e5 to d6, g8 to h8, a6 to a8,
a1 to e5, h8 to g8, e5 to f6, g7 to f6, a6 to a8,
a1 to e5, h8 to g8, e5 to f6, g8 to h8, a6 to a8,
a1 to e5, h8 to g8, e5 to f6, g8 to f8, a6 to a8,
Data sent to process 262177
received response from 262176, 7 tasks still running.
    a1 to d4, h8 to g8, a6 to a8,
    a1 to d4, h8 to g8, d4 to c5, g8 to h8, a6 to a8,
    a1 to d4, h8 to g8, d4 to f6, g7 to f6, a6 to a8,
    a1 to d4, h8 to g8, d4 to f6, g8 to h8, a6 to a8,
    a1 to d4, h8 to g8, d4 to f6, g8 to f8, a6 to a8,
Data sent to process 262176
received response from 262179, 7 tasks still running.
    a6 to a4, h8 to g8, a1 to f6, g7 to f6, a4 to a8,
    a6 to a4, h8 to g8, a1 to f6, g8 to h8, a4 to a8,
    a6 to a4, h8 to g8, a1 to f6, g8 to f8, a4 to a8,
    a6 to a4, h8 to g8, a4 to a8,
    a6 to a4, h8 to g8, a4 to f4, g8 to h8, f4 to f8,
Data sent to process 262179
received response from 262180, 7 tasks still running.
    a6 to a3, h8 to g8, a1 to f6, g7 to f6, a3 to a8,
    a6 to a3, h8 to g8, a1 to f6, g8 to h8, a3 to a8,
    a6 to a3, h8 to g8, a1 to f6, g8 to f8, a3 to a8,
    a6 to a3, h8 to g8, a3 to a8,
    a6 to a3, h8 to g8, a3 to f3, g8 to h8, f3 to f8,
Data sent to process 262180
received response from 262178, 7 tasks still running.
    a1 to f6, g7 to f6, a6 to a7, f6 to f5, a7 to a8,
    a1 to f6, g7 to f6, a6 to a7, h8 to g8, a7 to a8,
    a1 to f6, g7 to f6, a6 to a8,
    a1 to f6, g7 to f6, a6 to a5, f6 to f5, a5 to a8,
    a1 to f6, g7 to f6, a6 to a5, h8 to g8, a5 to a8,
    a1 to f6, g7 to f6, a6 to a4, f6 to f5, a4 to a8,
    a1 to f6, g7 to f6, a6 to a4, h8 to g8, a4 to a8,
    a1 to f6, g7 to f6, a6 to a3, f6 to f5, a3 to a8,
    a1 to f6, g7 to f6, a6 to a3, h8 to g8, a3 to a8,
    a1 to f6, g7 to f6, a6 to a2, f6 to f5, a2 to a8,
    a1 to f6, g7 to f6, a6 to a2, h8 to g8, a2 to a8,
    a1 to f6, g7 to f6, a6 to a1, f6 to f5, a1 to a8,
a1 to f6, g7 to f6, a6 to a1, h8 to g8, a1 to a8,
a1 to f6, g7 to f6, a6 to b6, f6 to f5, b6 to b8,
a1 to f6, g7 to f6, a6 to b6, h8 to g8, b6 to b8,
a1 to f6, g7 to f6, a6 to c6, f6 to f5, c6 to c8,
a1 to f6, g7 to f6, a6 to c6, h8 to g8, c6 to c8,
a1 to f6, g7 to f6, a6 to d6, f6 to f5, d6 to d8,
a1 to f6, g7 to f6, a6 to d6, h8 to g8, d6 to d8,
a1 to f6, g7 to f6, a6 to e6, f6 to f5, e6 to e8,
a1 to f6, g7 to f6, a6 to e6, h8 to g8, e6 to e8,
a1 to f6, h8 to g8, a6 to a7, g7 to f6, a7 to a8,
a1 to f6, h8 to g8, a6 to a7, g8 to h8, a7 to a8,
a1 to f6, h8 to g8, a6 to a7, g8 to f8, a7 to a8,
a1 to f6, h8 to g8, a6 to a8,
a1 to f6, h8 to g8, a6 to a5, g7 to f6, a5 to a8,
a1 to f6, h8 to g8, a6 to a5, g8 to h8, a5 to a8,
a1 to f6, h8 to g8, a6 to a5, g8 to f8, a5 to a8,
a1 to f6, h8 to g8, a6 to a4, g7 to f6, a4 to a8,
a1 to f6, h8 to g8, a6 to a4, g8 to h8, a4 to a8,
a1 to f6, h8 to g8, a6 to a4, g8 to f8, a4 to a8,
a1 to f6, h8 to g8, a6 to a3, g7 to f6, a3 to a8,
a1 to f6, h8 to g8, a6 to a3, g8 to h8, a3 to a8,
a1 to f6, h8 to g8, a6 to a3, g8 to f8, a3 to a8,
a1 to f6, h8 to g8, a6 to a2, g7 to f6, a2 to a8,
a1 to f6, h8 to g8, a6 to a2, g8 to h8, a2 to a8,
a1 to f6, h8 to g8, a6 to a2, g8 to f8, a2 to a8,
a1 to f6, h8 to g8, a6 to a1, g7 to f6, a1 to a8,
a1 to f6, h8 to g8, a6 to a1, g8 to h8, a1 to a8,
a1 to f6, h8 to g8, a6 to a1, g8 to f8, a1 to a8,
a1 to f6, h8 to g8, a6 to a1, g8 to f8, a1 to a8,
Data sent to process 262178
received response from 262174, 7 tasks still running.

Data sent to process 262181, 7 tasks still running.

No solution found. received response from 262177, 4 tasks still running.

Data sent to process 262180, 3 tasks still running.
g6 to f7, h8 to h7, a6 to g6, h7 to h8, g6 to h6,
received response from 262178, 2 tasks still running.
No solution found. received response from 262174, 1 tasks still running.
No solution found. received response from 262181, 0 tasks still running.
No solution found. Terminating process 262174
Terminating process 262175
Terminating process 262176
Terminating process 262177
Terminating process 262178
Terminating process 262179
Terminating process 262180
Terminating process 262181

- - - - - - k  1 0 0 0 0 0 0 0  0 0 0 0 0 0 1 0
- - - - - - p -  1 0 0 0 1 2 1 0 0 0 0 0 1 1
R - - - - - K -  0 1 1 1 1 3 1 1 0 0 0 0 1 0 1
- - - - - -  1 0 0 0 1 1 1 1 0 0 0 0 0 0 0
- - - - - -  1 0 0 1 0 0 0 0 0 0 0 0 0 0 0
- - - - - -  1 0 1 0 0 0 0 0 0 0 0 0 0 0 0
- - - - - -  1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
B - - - - - -  1 0 0 0 0 0 0 0 0 0 0 0 0 0 0

Solution List:
a6 to a8,
a1 to g7, h8 to g8, a6 to a8,
a1 to g7, h8 to g8, g7 to h6, g8 to h8, a6 to a8,
a1 to g7, h8 to g8, g7 to f6, g8 to f8, a6 to a8,
a6 to a7, h8 to g8, a1 to f6, g7 to f6, a7 to a8,
a6 to a7, h8 to g8, a1 to f6, g8 to h8, a7 to a8,
a6 to a7, h8 to g8, a1 to f6, g8 to f8, a7 to a8,
a6 to a7, h8 to g8, a7 to a8,
a6 to a7, h8 to g8, a7 to f7, g8 to h8, f7 to f8,
a1 to b2, h8 to g8, a6 to a8,
a1 to b2, h8 to g8, b2 to a3, g8 to h8, a6 to a8,
a1 to b2, h8 to g8, b2 to f6, g7 to f6, a6 to a8,
a1 to b2, h8 to g8, b2 to f6, g8 to h8, a6 to a8,
a1 to b2, h8 to g8, b2 to f6, g8 to f8, a6 to a8,
a6 to a5, h8 to g8, a1 to f6, g7 to f6, a5 to a8,
a6 to a5, h8 to g8, a1 to f6, g8 to h8, a5 to a8,
a6 to a5, h8 to g8, a1 to f6, g8 to f8, a5 to a8,
a6 to a5, h8 to g8, a5 to a8,
a6 to a5, h8 to g8, a5 to f5, g8 to h8, f5 to f8,
a1 to c3, h8 to g8, a6 to a8,
a1 to c3, h8 to g8, c3 to b4, g8 to h8, a6 to a8,
a1 to c3, h8 to g8, c3 to f6, g7 to f6, a6 to a8,
a1 to c3, h8 to g8, c3 to f6, g8 to h8, a6 to a8,
a1 to c3, h8 to g8, c3 to f6, g8 to f8, a6 to a8,
a1 to e5, h8 to g8, a5 to a8,
a1 to e5, h8 to g8, e5 to d6, g8 to h8, a6 to a8,
a1 to e5, h8 to g8, e5 to f6, g8 to h8, a6 to a8,
a1 to e5, h8 to g8, e5 to f6, g8 to f8, a6 to a8,
a1 to d4, h8 to g8, a6 to a8,
a1 to d4, h8 to g8, d4 to c5, g8 to h8, a6 to a8,
a1 to d4, h8 to g8, d4 to f6, g7 to f6, a6 to a8,
a1 to d4, h8 to g8, d4 to f6, g8 to h8, a6 to a8,
a1 to d4, h8 to g8, d4 to f6, g8 to f8, a6 to a8,
a6 to a4, h8 to g8, a1 to f6, g7 to f6, a4 to a8,
a6 to a4, h8 to g8, a1 to f6, g8 to h8, a4 to a8,
a6 to a4, h8 to g8, a1 to f6, g8 to f8, a4 to a8,
a6 to a4, h8 to g8, a4 to a8,
a6 to a4, h8 to g8, a4 to f4, g8 to h8, f4 to f8,
a6 to a3, h8 to g8, a1 to f6, g7 to f6, a3 to a8,
a6 to a3, h8 to g8, a1 to f6, g8 to h8, a3 to a8,
a6 to a3, h8 to g8, a1 to f6, g8 to f8, a3 to a8,
a6 to a3, h8 to g8, a3 to a8,
a6 to a3, h8 to g8, a3 to f3, g8 to h8, f3 to f8,
a1 to f6, g7 to f6, a6 to a7, f6 to f5, a7 to a8,
a1 to f6, g7 to f6, a6 to a7, h8 to g8, a7 to a8,
a1 to f6, g7 to f6, a6 to a8,
a1 to f6, g7 to f6, a6 to a5, f6 to f5, a5 to a8,
a1 to f6, g7 to f6, a6 to a5, h8 to g8, a5 to a8,
a1 to f6, g7 to f6, a6 to a4, f6 to f5, a4 to a8,
a1 to f6, g7 to f6, a6 to a4, h8 to g8, a4 to a8,
a1 to f6, g7 to f6, a6 to a3, f6 to f5, a3 to a8,
a1 to f6, g7 to f6, a6 to a3, h8 to g8, a3 to a8,
a1 to f6, g7 to f6, a6 to a2, f6 to f5, a2 to a8,
a1 to f6, g7 to f6, a6 to a2, h8 to g8, a2 to a8,
a1 to f6, g7 to f6, a6 to a1, f6 to f5, a1 to a8,
a1 to f6, g7 to f6, a6 to a1, h8 to g8, a1 to a8,
a1 to f6, g7 to f6, a6 to a1, h8 to g8, a1 to a8,
a1 to f6, g7 to f6, a6 to a6, f6 to f5, b6 to b8,
a1 to f6, g7 to f6, a6 to b6, f6 to f5, b6 to b8,
a1 to f6, g7 to f6, a6 to b6, h8 to g8, b6 to b8,
a1 to f6, g7 to f6, a6 to c6, f6 to f5, c6 to c8,
a1 to f6, g7 to f6, a6 to c6, h8 to g8, c6 to c8,
a1 to f6, g7 to f6, a6 to d6, f6 to f5, d6 to d8,
a1 to f6, g7 to f6, a6 to d6, h8 to g8, d6 to d8,
a1 to f6, g7 to f6, a6 to e6, f6 to f5, e6 to e8,
a1 to f6, g7 to f6, a6 to e6, h8 to g8, e6 to e8,
a1 to f6, h8 to g8, a6 to a7, g7 to f6, a7 to a8,
a1 to f6, h8 to g8, a6 to a7, g8 to h8, a7 to a8,
a1 to f6, h8 to g8, a6 to a7, g8 to f8, a7 to a8,
a1 to f6, h8 to g8, a6 to a8,
a1 to f6, h8 to g8, a6 to a5, g7 to f6, a5 to a8,
a1 to f6, h8 to g8, a6 to a5, g8 to h8, a5 to a8,
a1 to f6, h8 to g8, a6 to a5, g8 to f8, a5 to a8,
a1 to f6, h8 to g8, a6 to a4, g7 to f6, a4 to a8,
a1 to f6, h8 to g8, a6 to a4, g8 to h8, a4 to a8,
a1 to f6, h8 to g8, a6 to a4, g8 to f8, a4 to a8,
a1 to f6, h8 to g8, a6 to a3, g7 to f6, a3 to a8,
a1 to f6, h8 to g8, a6 to a3, g8 to h8, a3 to a8,
a1 to f6, h8 to g8, a6 to a3, g8 to f8, a3 to a8,
a1 to f6, h8 to g8, a6 to a2, g7 to f6, a2 to a8,
a1 to f6, h8 to g8, a6 to a2, g8 to h8, a2 to a8,
a1 to f6, h8 to g8, a6 to a2, g8 to f8, a2 to a8,
a1 to f6, h8 to g8, a6 to a1, g7 to f6, a1 to a8,
a1 to f6, h8 to g8, a6 to a1, g8 to h8, a1 to a8,
a1 to f6, h8 to g8, a6 to a1, g8 to f8, a1 to a8,
a1 to f6, h8 to g8, a6 to b6, g7 to f6, b6 to b8,
a1 to f6, h8 to g8, a6 to b6, g8 to h8, b6 to b8,
a1 to f6, h8 to g8, a6 to b6, g8 to f8, b6 to b8,
a1 to f6, h8 to g8, a6 to c6, g7 to f6, c6 to c8,
a1 to f6, h8 to g8, a6 to c6, g8 to h8, c6 to c8,
a1 to f6, h8 to g8, a6 to c6, g8 to f8, c6 to c8,
a1 to f6, h8 to g8, a6 to d6, g7 to f6, d6 to d8,
a1 to f6, h8 to g8, a6 to d6, g8 to h8, d6 to d8,
a1 to f6, h8 to g8, a6 to d6, g8 to f8, d6 to d8,
a1 to f6, h8 to g8, f6 to e7, g8 to h8, a6 to a8,
a1 to f6, h8 to g8, f6 to g5, g8 to h8, a6 to a8,
a1 to f6, h8 to g8, f6 to g5, g8 to f8, a6 to a8,
a1 to f6, h8 to g8, f6 to h4, g8 to h8, a6 to a8,
a6 to a2, h8 to g8, a1 to f6, g7 to f6, a2 to a8,
a6 to a2, h8 to g8, a1 to f6, g8 to h8, a2 to a8,
a6 to a2, h8 to g8, a1 to f6, g8 to f8, a2 to a8,
a6 to a2, h8 to g8, a2 to a8,
a6 to a2, h8 to g8, a2 to f2, g8 to h8, f2 to f8,
a6 to b6, h8 to g8, a1 to f6, g7 to f6, b6 to b8,
a6 to b6, h8 to g8, a1 to f6, g8 to h8, b6 to b8,
a6 to b6, h8 to g8, a1 to f6, g8 to f8, b6 to b8,
a6 to b6, h8 to g8, b6 to b8,
a6 to c6, h8 to g8, a1 to f6, g7 to f6, c6 to c8,
a6 to c6, h8 to g8, a1 to f6, g8 to h8, c6 to c8,
a6 to c6, h8 to g8, a1 to f6, g8 to f8, c6 to c8,
a6 to c6, h8 to g8, c6 to c8,
a6 to e6, h8 to g8, e6 to e8,
a6 to d6, h8 to g8, a1 to f6, g7 to f6, d6 to d8,
a6 to d6, h8 to g8, a1 to f6, g8 to h8, d6 to d8,
a6 to d6, h8 to g8, a1 to f6, g8 to f8, d6 to d8,
g6 to f7, h8 to h7, a6 to g6, h7 to h8, g6 to h6,

[crb4906@mealy]$ exit

exit

Script done on Thu May 20 13:00:40 1999

[File “input6.txt”]

7

5

1

WNc8 1 0
WNd4 1 0
WBh7 1 0
WBh6 1 0
WPa2 1 0
WPa5 1 0
WQd8 1 0
WKf1 1 0
BPa3 1 0
B Pc5 1 0
BPe5 1 0
BRe4 1 0
BKd3 1 0

Script started on Thu May 20 09:35:19 1999

~/pvm3/bin/[crb4906@amdahl]$ ~/pvm3/bin/LINUX/chess < input6.txt

remaining srch levels = 5
White's turn flag = 1
solve for white flag = 1
num_processes = 7

- - N Q - - - - 0 0 1 0 1 2 2 1 0 0 0 0 0 0 0 0 1 0 1 1 1 0 0 0
- - - - - - - B 1 0 1 1 2 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
- - - - - - - B 0 3 1 2 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1
P - p - p - - 1 1 0 1 0 2 2 0 0 0 0 1 0 0 0 1 1 0 1 0 1 1
- - N r - - - 0 0 0 1 1 1 0 1 0 1 4 1 2 1 1 0 0 0 1 1 0 0 0

remaining srch levels = 5
White's turn flag = 1
solve for white flag = 1
num_processes = 7
Data sent to process 786433
Data sent to process 1048577
Data sent to process 1310721
Data sent to process 1572865
Data sent to process 1835009
Data sent to process 2097153
Data sent to process 262147
sent initial tasks
received response from 1835009, 6 tasks still running.
No solution found. Data sent to process 1835009
received response from 2097153, 6 tasks still running.
No solution found. Data sent to process 2097153
received response from 1310721, 6 tasks still running.
No solution found. Data sent to process 1310721
received response from 262147, 6 tasks still running.
No solution found. Data sent to process 262147
received response from 1835009, 6 tasks still running.
No solution found. Data sent to process 1835009
received response from 2097153, 6 tasks still running.
No solution found. Data sent to process 2097153
received response from 1835009, 6 tasks still running.
No solution found. Data sent to process 1835009
received response from 786433, 6 tasks still running.
No solution found. Data sent to process 786433
received response from 1048577, 6 tasks still running.
No solution found. Data sent to process 1048577
received response from 1310721, 6 tasks still running.
No solution found. Data sent to process 1310721
received response from 2097153, 6 tasks still running.
No solution found. Data sent to process 2097153
received response from 1572865, 6 tasks still running.
No solution found. Data sent to process 1572865
received response from 1835009, 6 tasks still running.
No solution found. Data sent to process 1835009
received response from 1048577, 6 tasks still running.
No solution found. Data sent to process 1048577
received response from 1310721, 6 tasks still running.
No solution found. Data sent to process 1310721
received response from 2097153, 6 tasks still running.
No solution found. Data sent to process 2097153
received response from 1835009, 6 tasks still running.
No solution found. Data sent to process 1835009
received response from 786433, 6 tasks still running.
No solution found. Data sent to process 786433
received response from 262147, 6 tasks still running.
No solution found. Data sent to process 262147
received response from 2097153, 6 tasks still running.
d8 to g5, c5 to c4, c8 to b6, c4 to c3, q5 to e3,
d8 to g5, c5 to c4, c8 to b6, c4 to c3, q5 to d2,
d8 to g5, c5 to c4, c8 to b6, d3 to d4, q5 to e3,
d8 to g5, c5 to c4, c8 to b6, d3 to c3, q5 to d2,
d8 to g5, c5 to c4, c8 to b6, e5 to d4, q5 to d2,
d8 to g5, c5 to c4, q5 to e3,
d8 to g5, c5 to c4, q5 to d2,
d8 to g5, c5 to d4, g5 to g8, d3 to c2, q8 to b3,
d8 to g5, c5 to d4, g5 to g8, d3 to c3, g8 to b3,
d8 to g5, d3 to c4, c8 to b6, c4 to d4, q5 to e3,
d8 to g5, d3 to c4, c8 to b6, c4 to d3, g5 to d2,
d8 to g5, d3 to c4, c8 to b6, c4 to d3, q5 to e3,
d8 to g5, d3 to c4, c8 to b6, c4 to d3, q5 to d2,
d8 to g5, d3 to c4, c8 to b6, c4 to d3, q5 to d2,
d8 to g5, e5 to d4, g5 to g3, d3 to c4, e5 to b3,
d8 to g5, e5 to d4, g5 to c1, c5 to c4, c1 to d2,
Data sent to process 2097153
received response from 1835009, 6 tasks still running.
No solution found. Data sent to process 1835009
received response from 1310721, 6 tasks still running.
No solution found. Data sent to process 1310721
received response from 1572865, 6 tasks still running.
No solution found. Data sent to process 1572865
received response from 1048577, 6 tasks still running.
No solution found. Data sent to process 1048577
received response from 2097153, 6 tasks still running.
No solution found. Data sent to process 2097153
received response from 1310721, 6 tasks still running.
No solution found. Data sent to process 1310721
received response from 786433, 6 tasks still running.
No solution found. Data sent to process 786433
received response from 1835009, 6 tasks still running.
No solution found. Data sent to process 1835009
received response from 2097153, 6 tasks still running.
No solution found. Data sent to process 2097153
received response from 1310721, 6 tasks still running.
No solution found. Data sent to process 1310721
received response from 262147, 6 tasks still running.
No solution found. Data sent to process 262147
received response from 2097153, 6 tasks still running.
No solution found. Data sent to process 2097153
received response from 1310721, 6 tasks still running.
No solution found. Data sent to process 1310721
received response from 2097153, 5 tasks still running.
No solution found. received response from 786433, 4 tasks still running.
No solution found. received response from 1048577, 3 tasks still running.
No solution found. received response from 1572865, 2 tasks still running.
No solution found. received response from 1835009, 1 tasks still running.
No solution found. received response from 262147, 0 tasks still running.
No solution found. Terminating process 786433

Terminating process 1048577
Terminating process 1310721
Terminating process 1572865
Terminating process 1835009
Terminating process 2097153
Terminating process 262147

Solution List:

d8 to g5, c5 to c4, c8 to b6, c4 to c3, g5 to e3,
d8 to g5, c5 to c4, c8 to b6, c4 to c3, g5 to d2,
d8 to g5, c5 to c4, c8 to b6, d3 to d4, g5 to e3,
d8 to g5, c5 to c4, c8 to b6, d3 to c3, g5 to d2,
d8 to g5, c5 to c4, c8 to b6, e5 to d4, g5 to d2,
d8 to g5, c5 to c4, g5 to e3,
d8 to g5, c5 to c4, g5 to d2,
d8 to g5, c5 to d4, g5 to g8, d3 to c2, g8 to b3,
d8 to g5, c5 to d4, g5 to g8, d3 to c3, g8 to b3,
d8 to g5, d3 to c4, c8 to b6, c4 to d4, g5 to e3,
d8 to g5, d3 to c4, c8 to b6, c4 to d4, g5 to d2,
d8 to g5, d3 to c4, c8 to b6, c4 to d3, g5 to e3,
d8 to g5, d3 to c4, c8 to b6, c4 to d3, g5 to d2,
d8 to g5, d3 to c4, c8 to b6, c4 to c3, g5 to d2,
d8 to g5, d3 to c3, g5 to d2, c3 to c4, c8 to b6,
d8 to g5, d3 to c3, g5 to d2, c3 to c4, h7 to g8,
d8 to g5, e5 to d4, g5 to g3, d3 to c4, g3 to b3,
d8 to g5, e5 to d4, g5 to g3, d3 to c2, g3 to b3,
d8 to g5, e5 to d4, g5 to c1, c5 to c4, c1 to d2,

[crb4906@amdahl]$ exit

exit

Script done on Thu May 20 09:47:13 1999
8. Source Code

```c
/*
 * master.c
 */
#ifdef _WINDOWS
#include <stdlib.h>
#include <malloc.h>
#endif
#include <stdio.h>
#include "solve.h"
#include "board.h"
#include "slave.h"

#ifdef _WINDOWS
#include </home_mealy/crb4906/pvm3/include/pvm3.h>
#define SLAVENAME "chess_s"
#endif

/* Internal Functions */

void master( void );
void exit_program( SolutionNode **solution_list );
void receive_response( SolutionNode **solution_list );
boolean read_input_file( Problem *orig_problem, int *num_proc );
void fshow_problem( FILE *fp, Problem *p );
void log_problem_list( ProblemNode *list, char *fname );
void log_solution_list( SolutionNode *slist, char *fname );
void show_solutions( SolutionNode *slist );

#define MAX_BREAKDOWN_MOVES 2
#define TID_IDLE 0
#define TID_RUNNING 1

typedef struct {
    Problem   *problem;
    Move      move[MAX_BREAKDOWN_MOVES];
    void      *next;
} TaskNode;

void main()
{
    master();
}

void master( void )
{
    ProblemNode   *new_search_list;
    SearchList    search_list;
    SolutionNode  *solution_list=NULL;
    Problem       orig_problem;
    int           num_processes;
#ifdef _WINDOWS
    int mytid;
    int tid[MAX_PROCESSES];
    int tid_status[MAX_PROCESSES];
    int rcv_tid;
    int numtasks;
    int num_running_tasks=0;
    int i;
    int msgtag;
#endif
```
int message;
#endif

/* input the problem from a text file (including # of processes) */
if ( !read_input_file( &orig_problem, &num_processes ) ) return;
printf( "remaining srch levels = %d\n", orig_problem.remaining_srch_levels );
printf( "White's turn flag = %d\n", orig_problem.whites_turn_flag );
printf( "solve for white flag = %d\n", orig_problem.solve_for_white_flag );
if ( orig_problem.remaining_srch_levels < 1 ) {
  exit_program( &solution_list );
}
#endif

/* input the problem from a text file (including # of processes) */
if ( !read_input_file( &orig_problem, &num_processes ) ) return;
printf( "remaining srch levels = %d\n", orig_problem.remaining_srch_levels );
printf( "White's turn flag = %d\n", orig_problem.whites_turn_flag );
printf( "solve for white flag = %d\n", orig_problem.solve_for_white_flag );
if ( orig_problem.remaining_srch_levels < 1 ) {
  exit_program( &solution_list );
}
#endif

num_processes = min( num_processes, MAX_PROCESSES );
printf( "num_processes = %d\n", num_processes );
// Enroll in PVM
mytid = pvm_mytid();

// initialize the slave processes as idle
for( i=0; i < num_processes; i++ ) {
  tid_status[i] = TID_IDLE;
}

// start the slave tasks
numtasks = pvm_spawn( SLAVENAME, (char**)0, 0, "", num_processes, tid );
if ( numtasks < num_processes ) {
  printf( "Trouble spawning slaves, aborting\n" );
  for( i=0; i < numtasks; i++ ) {
    pvm_kill( tid[i] );
  }
  pvm_exit();
  exit(1);
}
#endif

// get the attacked pieces for the original problem
find_attacked_squares( &orig_problem );

// log the original problem
fshow_problem( stdout, &orig_problem );

/* break down the problem into a list of tasks */
new_search_list = break_down_problem( &orig_problem );
#endif

log_problem_list( new_search_list, "log.txt" );
#else
log_problem_list( new_search_list, "\chris\chessapp\log.txt" );
#endif

search_list.problem_start = new_search_list;
search_list.cur_problem = new_search_list;

// solve each of the problems individually
while( search_list.cur_problem != NULL ) {
  char fname[40];
  static int scnt=0;
  slave_process( search_list[cur_problem->problem, &solution_list );
  search_list[cur_problem = search_list.cur_problem->next;
  // report the results of the search
  scnt++;
  sprintf( fname, "\chris\chessapp\solution%d.txt", scnt );
  log_solution_list( solution_list, fname );
  // destroy the solution list
  destroy_solution_list( &solution_list );
}
#endif

// hand out the initial processes
msgtag = 0;
num_running_tasks = 0;

}
for( i=0; i < num_processes; i++ ) {
    if ( search_list.cur_problem == NULL ) break;
    pvm_initsend( PvmDataDefault );
    message = SOLVE_PROBLEM;
    pvm_pkint( &message, 1, 1 );
    pvm_pkbyte( (char*)search_list.cur_problem->problem, sizeof(Problem), 1 );
    pvm_mcast( &tid[i], 1, msgtag );
    printf( "Data sent to process %d\n", tid[i] );
    //tid_status[i] = TID_RUNNING;
    num_running_tasks++;
    search_list.cur_problem = search_list.cur_problem->next;
}
printf( "sent initial tasks\n" );

while( num_running_tasks > 0 ) {
    // Wait for a response.  Send another problem to that process if available.
    pvm_recv( -1, 5 );
    pvm_upkint( &rcv_tid, 1, 1 );
    num_running_tasks--;
    printf( "received response from %d, %d tasks still running.\n", rcv_tid, num_running_tasks );
    // get the result from the process
    receive_response( &solution_list );
    // send another task to this process if necessary
    if ( search_list.cur_problem != NULL ) {
        pvm_initsend( PvmDataDefault );
        pvm_pkint( &message, 1, 1 );
        pvm_pkbyte( (char*)search_list.cur_problem->problem, sizeof(Problem), 1 );
        pvm_mcast( &rcv_tid, 1, 0 );
        printf( "Data sent to process %d\n", rcv_tid );
        num_running_tasks++;
        search_list.cur_problem = search_list.cur_problem->next;
    }
}
printf( "Terminating processes\n" );
    pvm_initsend( PvmDataDefault );
    message = TERMINATE_PROCESS;
    pvm_pkint( &message, 1, 1 );
    pvm_mcast( tid, num_processes, msgtag );
*/
#endif

#ifndef _WINDOWS
for( i=0; i < num_processes; i++ ) {
    printf( "Terminating process %d\n", tid[i] );
    pvm_kill( tid[i] );
}
pvm_exit();

// show the original problem and the solutions
fshow_problem( stdout, &orig_problem );
show_solutions( solution_list );
#endif

/* free memory that was allocated and terminate the program */
exit_program( &solution_list );
}

/*
* Get the response from a task.
* Add the problem solution(s) to the list.
* Display the results.
*/
#endif

void receive_response( SolutionNode **solution_list )
{
    SolutionNode *newnode;
    int num_solutions;
    num_solutions = 0;
    *solution_list = NULL;

    newnode = (SolutionNode *)malloc(sizeof(SolutionNode));
    if (newnode == NULL) {
        printf("Memory allocation failed\n");
        exit(1);  // Exit the program if memory allocation failed
    }

    newnode->problem = (char*)malloc(sizeof(Problem));
    if (newnode->problem == NULL) {
        free(newnode);
        printf("Memory allocation failed\n");
        exit(1);  // Exit the program if memory allocation failed
    }

    newnode->solution = (char*)malloc(sizeof(SolutionNode));
    if (newnode->solution == NULL) {
        free(newnode->problem);
        free(newnode);
        printf("Memory allocation failed\n");
        exit(1);  // Exit the program if memory allocation failed
    }

    printf("Received solution from process %d\n", i);
    printf("Solution: %s\n", newnode->solution);
    printf("Problem: %s\n", newnode->problem);

    num_solutions++;
    *solution_list = newnode;
    return;

    free(newnode->solution);
    free(newnode->problem);
    free(newnode);
}

int main(int argc, char** argv)
{
    int num_processes = atoi(argv[1]);
    SolutionNode *solution_list = NULL;
    int i;

    // Initialize PETSc
    PetscInitialize(&argc, &argv, (char *)0, (char *)0);

    // Create a communicator for the processes
    PETSC_COMM_WORLD = PETSC_COMM_WORLD;

    // Create a group of processes
    PETSC_COMM_WORLD = PETSC_COMM_WORLD;

    // Create a communicator for each process
    PETSC_COMM_WORLD = PETSC_COMM_WORLD;

    // Create a solver for each process
    PETSC_COMM_WORLD = PETSC_COMM_WORLD;

    // Start the main loop
    while (num_solutions < num_processes) {
        // Wait for a response from a process
        receive_response( &solution_list );
        num_solutions++;
    }

    // Clean up PETSc
    PetscFinalize();

    return 0;
}
unsigned char num_moves;
int i, j;
pvm_upkint( &num_solutions, 1, 1 );
if ( num_solutions == 0 ) {
    printf( "No solution found. " );
}
for( i=0; i < num_solutions; i++ ) {
    // get the number of moves for this solution
    pvm_upkbyte( (char*)&num_moves, 1, 1 );
    // create a new solution node
    newnode = (SolutionNode*) malloc( sizeof( SolutionNode ));
    newnode->next = NULL;
    newnode->num_moves = num_moves;
    // add to the solution list
    add_solution( solution_list, newnode );
    for( j=0; j < num_moves; j++ ) {
        pvm_upkbyte( (char*)&newnode->move[j].src, 1, 1 );
        pvm_upkbyte( (char*)&newnode->move[j].dest, 1, 1 );
        printf( "%c%d to %c%d, ",
            newnode->move[j].src.col+'a', newnode->move[j].src.row+1,
            newnode->move[j].dest.col+'a', newnode->move[j].dest.row+1 );
    }
    printf( "\n" );
}
#endif
/*
 * Free any allocated memory and exit the program.
 */
void exit_program( SolutionNode **solution_list ) {
    SolutionNode *t1;
    /* free the solution list */
    t1 = *solution_list;
    while( t1 != NULL ) {
        t1 = t1->next;
        free( *solution_list );
        *solution_list = t1;
    }
    /* exit the program */
    exit(0);
}

/*
 * Read the problem to solve from a text input file.
 * The format is as follows:
 *   Line 1 : number of processors (decimal integer)
 *   Lines 2~ : Piece/Position (ex. WKe1)
 */
boolean read_input_file( Problem *p, int *num_proc ) {
    int i, j;
    char str[10];
    int has_moved_flag, en_passent_flag;
    FILE *fp;
    char *fname;
#ifdef _WINDOWS
    fname = "input.txt";
#endif
    /* open the input file */
    fp = fopen( fname, "r" );
    if ( !fp ) {
        printf( "can't open input file '%s', fname );
    }
```c
    *num_proc = 0;
    return FALSE;
}
#else
    fp = stdin;
#endif

/* read first line = num processors */
    fscanf( fp, "%d", num_proc );

/* read second line = search levels remaining */
    fscanf( fp, "%d", &p->remaining_srch_levels );

/* read third number = whites turn flag */
    fscanf( fp, "%d", &p->whites_turn_flag );
    p->solve_for_white_flag = p->whites_turn_flag;

/* initialize the board squares to unoccupied */
for( i=0; i < 8; i++ ) {
    for( j=0; j < 8; j++ ) {
        p->board[i][j].occupied = FALSE;
    }
}
while( !feof( fp ) ) {
    char      color;
    char      piece_type;
    char      prow;
    char      pcol;
    fscanf( fp, "%s", str );
    if ( str[0] == (char)NULL ) break;
    fscanf( fp, "%d", &has_moved_flag );
    fscanf( fp, "%d", &en_passent_flag );

    switch( str[0] ) {
    case 'W':
        color = WHITE;
        break;
    case 'B':
        color = BLACK;
        break;
    default:
        return FALSE;
        break;
    }

    switch( str[1] ) {
    case 'P':
        piece_type = PAWN;
        break;
    case 'N':
        piece_type = KNIGHT;
        break;
    case 'B':
        piece_type = BISHOP;
        break;
    case 'R':
        piece_type = ROOK;
        break;
    case 'Q':
        piece_type = QUEEN;
        break;
    case 'K':
        piece_type = KING;
        break;
    default:
        return FALSE;
        break;
    }
```
    pcol = str[2] - 'a';
    pcol = str[2] - 'A';
} else {
    return FALSE;
}

    return FALSE;
} else {
    prow = str[3] - '1';
}

p->board[prow][pcol].pcolor = color;
p->board[prow][pcol].ptype = piece_type;
p->board[prow][pcol].occupied = TRUE;
p->board[prow][pcol].has_moved = has_moved_flag;
p->board[prow][pcol].en_passent_flag = en_passent_flag;
}

return TRUE;
}

void log_problem_list( ProblemNode *list, char *fname )
{
    FILE *fptr;
    ProblemNode *temp = list;

    fptr = fopen( fname, "w" );
    if ( !fptr ) return;

    while( temp != NULL ) {
        fshow_problem( fptr, temp->problem );
        fprintf( fptr, "solve for white = %d
", temp->problem->solve_for_white_flag );
        fprintf( fptr, "white's turn = %d
", temp->problem->whites_turn_flag );
        fprintf( fptr, "move src = %c%d, dest = %c%d
", temp->problem->move.src.col+'a',

        temp->problem->move.src.row+1, temp->problem->move.dest.col+'a', temp->problem->move.dest.row+1 );
        if ( temp->problem->move.src2.row != -1 ) fprintf( fptr, "src2 = %c%d, dest2 = %c%d
", temp->problem->move.src2.col+'a', temp->problem->move.src2.row+1, temp->problem->move.dest2.col+'a', temp->problem->move.dest2.row+1 );
        if ( temp->problem->move.delete_loc.row != -1 ) fprintf( fptr, "delete_loc = %c%d
", temp->problem->move.delete_loc.col+'a', temp->problem->move.delete_loc.row+1 );
        if ( temp->problem->move.new_piece_loc.row != -1 ) fprintf( fptr, "new_piece = %d, new_piece_loc = %c%d
", temp->problem->move.new_piece, temp->problem->move.new_piece_loc.col+'a', temp->problem->move.new_piece_loc.row+1 );
        fprintf( fptr, "remaining srch levels = %d
", temp->problem->remaining_srch_levels );
        fprintf( fptr, "\n\n\n" );
        temp = temp->next;
    }

    fclose( fptr );
}

void fshow_problem( FILE *fp, Problem *p )
{
    int i, j;
    char pchar;

    for( i=-7; i >= 0; i-- ) {
        for( j=0; j < 8; j++ ) {
            pchar = '-';
            if ( p->board[i][j].occupied ) {
                switch( p->board[i][j].ptype ) {
                case 1:
                    pchar = 'K';
                    break;
                case 2:
                    pchar = 'Q';
                    break;
                case 3:
                    pchar = 'R';
                    break;
                case 4:
                    pchar = 'N';
                    break;
                case 5:
                    pchar = 'B';
                    break;
                case 6:
                    pchar = 'P';
                    break;
                case 7:
                    pchar = 'p';
                    break;
                case 8: 
                    pchar = 'n';
                    break;
                default:
                    pchar = ' ';
                    break;
                }
            }
            fprintf( fp, "%c", pchar );
        }
        fprintf( fp, "\n\n" );
    }
}
case PAWN:
    pchar = 'p';
    break;
case KNIGHT:
    pchar = 'n';
    break;
case BISHOP:
    pchar = 'b';
    break;
case ROOK:
    pchar = 'r';
    break;
case QUEEN:
    pchar = 'q';
    break;
case KING:
    pchar = 'k';
    break;
default:
    /* software error */
    fprintf( fp, "software error: ptype = %d\n", p->board[i][j].ptype );
    break;
}
if ( p->board[i][j].pcolor == WHITE ) {
    pchar += 'A' - 'a';
}
/*
    fprintf( fp, "    ");
    for( j=0; j < 8; j++ ) {
        pchar = p->board[i][j].has_moved+'0';
        fprintf( fp, "%c ", pchar );
    }
*/
fprintf( fp, "    ");
for( j=0; j < 8; j++ ) {
    pchar = p->board[i][j].attacked_by_white+'0';
    fprintf( fp, "%c ", pchar );
}
for( j=0; j < 8; j++ ) {
    pchar = p->board[i][j].attacked_by_black+'0';
    fprintf( fp, "%c ", pchar );
}
/*
    fprintf( fp, "    ");
    for( j=0; j < 8; j++ ) {
        pchar = p->board[i][j].has_moved+'0';
        fprintf( fp, "%c ", pchar );
    }
*/
fprintf( fp, "\n" );
}

void log_solution_list( SolutionNode *slist, char *fname )
{
    FILE *fp;
    int i;
    SolutionNode *temp=slist;

    if (fp = fopen( fname, "w" )) {
        while( temp != NULL ) {
            for( i=0; i < temp->num_moves; i++ ) {
                fprintf( fp, "%c%d to %c%d, ", temp->move[i].src.col+1, temp->move[i].src.row+1, temp->move[i].dest.col+1, temp->move[i].dest.row+1 );
            }
            fprintf( fp, "\n" );
            temp = temp->next;
        }
    fclose( fp );
}
void show_solutions( SolutionNode *slist )
{
    SolutionNode *temp=slist;
    int i;

    printf( "\n\nSolution List:\n\n" );

    if ( temp == NULL ) printf( "No solution found.\n" );
    while( temp != NULL ) {
        for( i=0; i < temp->num_moves; i++ ) {
            printf( "%c%d to %c%d,\n", temp->move[i].src.col+'a', temp->move[i].src.row+1,
                     temp->move[i].dest.col+'a', temp->move[i].dest.row+1 );
        }
        printf( "\n" );
        temp = temp->next;
    }
}

#pragma once

#ifdef _WINDOWS
#include <stdlib.h>
#include <malloc.h>
#else
#include </home_mealy/crb4906/pvm3/include/pvm3.h>
#endif
#include <stdio.h>
#include "solve.h"
#include "slave.h"
#include "pieces.h"
#include "board.h"
#include "movelist.h"

#endif

/*
 * Internal Functions
 */
#ifndef _WINDOWS
void slave_process( void );
void solve_problem( int master_tid, int mytid );
#endif

/*
Slave Process Function:
Receive a given problem and put it into the top of the problem search list.
Repeatedly run the solving function until the problem has been solved.
*/
#ifndef _WINDOWS
void main()
{
    slave_process();
}

void slave_process( void )
{
    int mytid;
    int master_tid;
    int mt;
    int message;
    boolean done=FALSE;

    /* register with pvm */
    mytid = pvm_mytid();
    master_tid = pvm_parent();

    slave_process();
    while( !done ) {
        slave_process();
    }
}
#endif
while(!done) {
    /* wait for a message from the master */
    mt = 0;
    message = NO_MESSAGE;
    pvm_nrecv(-1, mt);
    pvm_upkint(&message, 1, 1);

    switch(message) {
    case SOLVE_PROBLEM:
        solve_problem(master_tid, mytid);
        break;
    case TERMINATE_PROCESS:
        // the program is complete - terminate this process
        // done = TRUE;
        break;
    case TRANSFER_WORK:
        // transfer some work to another process
        break;
    case NO_MESSAGE:
        default:
            break;
    }
}

/* terminate the process */
pvm_exit();
}

#else
void slave_process(Problem *orig_problem, SolutionNode **solution_list)
{
    ProblemNode *orig_problem_node;
    SearchList search_list[MAX_SEARCH_LEVEL];
    int search_level = 0;
    // ProblemResult solution_result=NO_SOLUTION;

    // create the initial problem node
    // (note: this node will be destroyed in the solving process)
    orig_problem_node = (ProblemNode*)malloc(sizeof(ProblemNode));
    orig_problem_node->problem = orig_problem;
    orig_problem_node->next = NULL;

    // initialize the variables used by the solving routine
    search_level = 0;
    search_list[0].problem_start = orig_problem_node;
    search_list[0].cur_problem = orig_problem_node;

    // solve the problem
    while(!solve(solution_list, search_list, &search_level)) {};

    // note: the original problem node will be destroyed in the solving process
}
#endif

#ifndef _WINDOWS
void solve_problem(int master_tid, int mytid)
{
    Problem orig_problem;
    ProblemNode *orig_problem_node;
    SolutionNode *solution_list=NULL;
    SearchList search_list[MAX_SEARCH_LEVEL];
    int search_level = 0;
    int mt;

    // get the problem from the master
    pvm_upkbyte((char*)&orig_problem, sizeof(Problem), 1);

// create the initial problem node
// (note: this node will be destroyed in the solving process)
orig_problem_node = (ProblemNode*) malloc( sizeof( ProblemNode ));
orig_problem_node->problem = &orig_problem;
orig_problem_node->next = NULL;

// initialize the variables used by the solving routine
search_level = 0;
search_list[0].problem_start = orig_problem_node;
search_list[0].cur_problem = orig_problem_node;

/* run the solver routine until the problem is complete */
while( !solve( &solution_list, search_list, &search_level )) {
}

/* send the solution list to the master process */
{
    int i;
    SolutionNode *temp=solution_list;
    int num_solutions=0;
    // count the number of solutions
    while( temp != NULL ) {
        num_solutions++;
        temp = temp->next;
    }
    pvm_initsend( PvmDataDefault );
    pvm_pkint( &mytid, 1, 1 );
    pvm_pkint( &num_solutions, 1, 1 );
    // add the solution move lists
    temp = solution_list;
    while( temp != NULL ) {
        pvm_pkbyte( (char*)&temp->num_moves, 1, 1 );
        for( i=0; i < temp->num_moves; i++ ) {
            pvm_pkbyte( (char*)&temp->move[i].src, 1, 1 );
            pvm_pkbyte( (char*)&temp->move[i].dest, 1, 1 );
        }
        temp = temp->next;
    }
}

// send the solutions to the master
mt = 5;
pvm_send( master_tid, mt );

/* destroy the solution list */
destroy_solution_list( &solution_list );
}
#endif

/*
*solve.c
*/
#endif _WINDOWS
#include <stdlib.h>
#include <malloc.h>
#include <stdio.h>
#include "solve.h"
#include "slave.h"
#include "pieces.h"
#include "board.h"
#include "movelist.h"

/*
* Internal Functions
*/
void add_to_solution_list( SolutionNode **solution_list,
    SearchList search_list[MAX_SEARCH_LEVEL],
    int search_level );

void remove_from_solution_list( SolutionNode **solution_list,
    SearchList search_list[MAX_SEARCH_LEVEL],
    int search_level );

ProblemNode* create_problem_node( Problem *p );

ProblemResult check_for_solution( Problem *p );

boolean is_checkmate( Problem *p );

boolean is_stalemate( Problem *p );

void add_to_problem_list( ProblemNode **list, ProblemNode *newnode );

ProblemNode* create_new_problem( Problem *p, Move *move );

void copy_problem( Problem *dest, Problem *src );

void move_piece( Problem *p, PieceLoc src, PieceLoc dest );

void perform_move( Problem *p, Move *m );

void destroy_search_list( SearchList search_list[MAX_SEARCH_LEVEL],
    int srch_level );

ProblemResult get_search_list_results( boolean solving_players_turn,
    SearchList search_list[MAX_SEARCH_LEVEL],
    int search_level );

boolean solve( SolutionNode **solution_list,
    SearchList search_list[MAX_SEARCH_LEVEL], int *search_level )
{
    ProblemResult status;
    ProblemNode *pnode;
    Problem *p;

    /* get a pointer to the current problem */
    pnode = search_list[*search_level].cur_problem;

    //
    // If the current problem is null, then:
    // - the end of the current search list has been reached
    // - if already at highest search level then problem is complete
    // - analyze the search list for possible solutions
    // - mark the next higher level problem accordingly
    // - discard the current search level
    // Otherwise:
    // - continue analyzing the current search list
    //
    if ( pnode == NULL ) {
        ProblemResult result;
        boolean solving_players_turn=FALSE;

        // get the results of the current search
        p = search_list[*search_level].problem_start->problem;
        if ( p->whites_turn_flag != p->solve_for_white_flag ) {
            solving_players_turn = TRUE;
        }
        result = get_search_list_results( solving_players_turn, search_list, *search_level
        );

        // destroy the current search list
        destroy_search_list( search_list, *search_level );
        search_list[*search_level].problem_start = NULL;
        search_list[*search_level].cur_problem = NULL;
    }
    result = get_search_list_results( solving_players_turn, search_list, *search_level
    );

    // if ( result == NO_SOLUTION ) {
        remove_from_solution_list( solution_list, search_list, *search_level-1 );
    }
    if ( *search_level == 0 ) {
        ProblemResult result;
        boolean solving_players_turn=FALSE;

        // get the results of the current search
        p = search_list[*search_level].problem_start->problem;
        if ( p->whites_turn_flag != p->solve_for_white_flag ) {
            solving_players_turn = TRUE;
        }
        result = get_search_list_results( solving_players_turn, search_list, *search_level
        );

        // destroy the current search list
        destroy_search_list( search_list, *search_level );
        search_list[*search_level].problem_start = NULL;
        search_list[*search_level].cur_problem = NULL;
    }
// the problem is complete
// solution_result = result;
return TRUE;
} else {
// continue solving at the next higher level
(*search_level)--;
search_list[*search_level].cur_problem->result = result;
search_list[*search_level].cur_problem =
    search_list[*search_level].cur_problem->next;
    return FALSE;
}

/* get a ptr to the current problem being examined */
p = pnode->problem;

/* check to see if the current problem is completed */
status = check_for_solution( p );
switch( status ) {
    case POSSIBLE_SOLUTION:
        /* The current move line represents a possible solution. */
        * Mark the problem as such, add the move to the solution list
        * and continue with the rest of the current search level.
        */
        pnode->result = POSSIBLE_SOLUTION;
        // add this move to the problems solution list
        add_to_solution_list( solution_list, search_list, *search_level );
        search_list[*search_level].cur_problem = pnode->next;
        break;
    case NO_SOLUTION:
        /* The current move line is an invalid solution to the problem.
        * Mark the current problem as such and continue with the rest
        * of the current search level.
        */
        pnode->result = NO_SOLUTION;
        search_list[*search_level].cur_problem = pnode->next;
        // // remove any possible solutions that have been generated
        // under this move line.
        // //remove_from_solution_list( solution_list );
        break;
    case NOT_COMPLETED:
        default:
        /* The current move must be investigated further.
        * Create a deeper search line.
        */
        {
        ProblemNode *new_list;
        new_list = break_down_problem( p );
        /* NOTE: new_list should never be null */
        /* increment our search level */
        // If the new list is null, then a stalemate has been found.
        // This is not a valid solution.
        if ( new_list == NULL ) {
        pnode->result = NO_SOLUTION;
        search_list[*search_level].cur_problem = pnode->next;
        } else {
        (*search_level)++;
        search_list[*search_level].problem_start = new_list;
        search_list[*search_level].cur_problem = new_list;
        }
        break;
        }

    return FALSE;
}
/ Add the current move line to the list of possible solutions.

/**
 * void add_to_solution_list( SolutionNode ** solution_list,
 *                             SearchList search_list[MAX_SEARCH_LEVEL],
 *                             int search_level )
 *
 * SolutionNode *newnode;
 * int i;
 *
 * // create a new solution node
 * newnode = (SolutionNode *) malloc( sizeof( SolutionNode ));
 * newnode->next = NULL;
 * newnode->num_moves = search_level+1;
 * for( i=0; i < MAX_SEARCH_LEVEL; i++ ) {
 *     newnode->move[i].src.row = -1;
 *     newnode->move[i].src.col = -1;
 *     newnode->move[i].dest.row = -1;
 *     newnode->move[i].dest.col = -1;
 * }
 *
 * // put the current list of moves into the new solution
 * for( i=0; i <= search_level; i++ ) {
 *     copy_move( &newnode->move[i], &search_list[i].cur_problem->problem->move );
 * }
 *
 * // add to the list of possible solutions
 * add_solution( solution_list, newnode );
 */

/**
 * void remove_from_solution_list( SolutionNode ** solution_list,
 *                                 SearchList search_list[MAX_SEARCH_LEVEL],
 *                                 int search_level )
 *
 * SolutionNode *temp=*solution_list;
 * SolutionNode *last=NULL;
 * boolean match;
 * int i;
 *
 * if ( *solution_list != NULL ) {
 *     match = TRUE;
 * }
 *
 * while( temp != NULL ) {
 *     match = TRUE;
 *     for( i=0; i <= search_level; i++ ) {
 *         if ( temp->move[i].src.row != -1 &&
 *             search_list[i].cur_problem != NULL ) {
 *             if ( temp->move[i].src.row != search_list[i].cur_problem->problem->move.src.row ||
 *                 temp->move[i].src.col != search_list[i].cur_problem->problem->move.src.col ||
 *                 temp->move[i].dest.row != search_list[i].cur_problem->problem->move.dest.row ||
 *                 temp->move[i].dest.col != search_list[i].cur_problem->problem->move.dest.col ) {
 *                 match = FALSE;
 *                 break;
 *             }
 *         }
 *     }
 *     if ( match ) {
 *         if ( last == NULL ) {
 *             *solution_list = temp->next;
 *             free( temp );
 *             temp = *solution_list;
 *         } else {
 *             last->next = temp->next;
 *             free( temp );
 *             temp = *solution_list;
 *         }
 *     }
 *     last = temp;
 *     temp = temp->next;
 * }
 */
} else {
    last->next = temp->next;
    free( temp );
    temp = last->next;
} }
} else {
    last = temp;
    temp = temp->next;
}
}

ProblemResult get_search_list_results( boolean solving_players_turn,
          SearchList search_list[MAX_SEARCH_LEVEL],
          int search_level )
{
    ProblemNode *temp;
    ProblemResult result;
    temp = search_list[search_level].problem_start;
    if ( solving_players_turn ) {
        result = NO_SOLUTION;
        while( temp != NULL ) {
            if ( temp->result == POSSIBLE_SOLUTION ) {
                result = POSSIBLE_SOLUTION;
            }
            temp = temp->next;
        }
    } else {
        result = POSSIBLE_SOLUTION;
        while( temp != NULL ) {
            if ( temp->result == NO_SOLUTION ) {
                result = NO_SOLUTION;
            }
            temp = temp->next;
        }
    }
    return result;
}

#ifdef not_defined
void create_move_list( void )
{
    SolutionNode *newnode;
    int movecnt;

    /* create the new solution node */
    newnode = (SolutionNode*) malloc( sizeof(SolutionNode ));
    //newnode = new SolutionNode;
    newnode->next = NULL;

    /* get the current moves being analyzed */
    for( movecnt=0; movecnt <= search_level; movecnt++ ) {
        newnode->move[movecnt] = search_list[movecnt].cur_problem->problem->move;
    }

    /* append the new solution to the list */
    add_solution( solution_list, newnode );
}
#endif
/* Add a solution to the list. */
void add_solution( SolutionNode **solution_list, SolutionNode *newnode )
{
    if ( *solution_list == NULL ) {
        /* add the first node to the list */
        *solution_list = newnode;
        newnode->next = NULL;
    } else {
        /* find the last node in the list */
        SolutionNode *temp = *solution_list;
        while( temp->next != NULL ) {
            temp = temp->next;
        }
        /* append the node to the end of the list */
        temp->next = newnode;
        newnode->next = NULL;
    }
}

/* From the given problem, create a new problem node destined for a linked list. */
ProblemNode* create_problem_node( Problem *p )
{
    ProblemNode *newnode;
    //newnode = new ProblemNode;
    newnode = (ProblemNode*) malloc( sizeof( ProblemNode ));
    newnode->result = NOT_COMPLETED;
    newnode->next = NULL;
    newnode->problem = p;
    return newnode;
}

/* Check the given problem for the solution status.
* Determine if the node represents a valid solution, an invalid solution,
* or requires more analysis.
* Note that a solution is only possible if the player we're solving the
* problem for is not the player to move. */
ProblemResult check_for_solution( Problem *p )
{
    ProblemResult result;
    /* If the player to move is not the one we're solving for, then look
     * at the game status: checkmate indicates a possible solution and
     * stalemate indicates an incorrect solution, but an unresolved situation
     * must be examined using move search levels.
     */
    if ( p->solve_for_white_flag != p->whites_turn_flag ) {
        /* it's not the solving players turn to move */
        if ( is_checkmate( p ) ) {
            /* a valid solution line has been found */
            result = POSSIBLE_SOLUTION;
        } else if ( is_stalemate( p ) ) {
            /* an invalid solution has been found */
            result = NO_SOLUTION;
        } else if ( p->remaining_srch_levels > 0 ) {
            /* more investigation is required */
            result = NOT_COMPLETED;
        } else {
            /* ran out of search levels without finding a solution */
            result = NO_SOLUTION;
        }
    } else {
        /* it's the solving players turn to move */
    }
}
if (is_checkmate( p ) || is_stalemate( p ) ) {
    /*
    * The wrong player has been checkmated or stalemated.
    * This is an invalid solution to the problem.
    */
    result = NO_SOLUTION;
} else if ( p->remaining_search_levels > 0 ) {
    /* more investigation is required */
    result = NOT_COMPLETED;
} else {
    /* ran out of search levels without finding a solution */
    result = NO_SOLUTION;
}
}
return result;

/*
* Does the given problem represent a checkmate situation?
* Check to see if either king has any valid moves.
* If not and his square is also attacked, then it's mate.
*/
boolean is_checkmate( Problem *p ) {
    int i,j;
    unsigned char color;
    PieceLoc loc;
    boolean mate=FALSE;
    MoveList *mlist=NULL;
    // find the attacked squares first
    find_attacked_squares( p );
    // find the kings on the board
    for( i=0; i < 8; i++ ) {
        for( j=0; j < 8; j++ ) {
            if ( p->board[i][j].occupied && p->board[i][j].ptype == KING ) {
                color = p->board[i][j].pcolor;
                loc.row = i;
                loc.col = j;
                get_valid_moves( &mlist, p, loc );
                if ( mlist == NULL ) {
                    if ( color == WHITE ) {
                        if ( p->board[i][j].attacked_by_black )
                            mate = TRUE;
                    } else {
                        if ( p->board[i][j].attacked_by_white )
                            mate = TRUE;
                    }
                }
                destroy_movelist( &mlist );
            }
        }
    }
    return mate;
}
/*
* Break down the given problem into a list of new problems using
* all possible moves for every piece of the current player from
* the initial position.
*/
ProblemNode* break_down_problem( Problem *p ) {
    int row,col;
    MoveList *movelist=NULL;
    MoveList *temp;
    ProblemNode *problem_list=NULL;
    /*
    * For every piece on the board of the current players color,
* create a new problem for all possible moves.
*/
for( col=0; col < 8; col++ ) {
    for( row=0; row < 8; row++ ) {
        if ( p->board[row][col].occupied &&
            p->board[row][col].pcolor == p->whites_turn_flag ) {
            PieceLoc ploc;
            ploc.col = col;
            ploc.row = row;
            get_valid_moves( &movelist, p, ploc );
            temp = movelist;
            while( temp != NULL ) {
                ProblemNode *newnode;
                newnode = create_new_problem( p, &temp->move );
                add_to_problem_list( &problem_list, newnode );
                temp = temp->next;
            }
            destroy_movelist( &movelist );
        }
    }
}
return problem_list;
}

/*
 * Add a node to a given problem list.
*/
void add_to_problem_list( ProblemNode **list, ProblemNode *newnode )
{
    if ( *list == NULL ) {
        *list = newnode;
        newnode->next = NULL;
    } else {
        ProblemNode *t;
        t = *list;
        while( t->next != NULL ) {
            t = t->next;
        }
        t->next = newnode;
        newnode->next = NULL;
    }
}

/*
 * Create a new problem from a given problem and move combination.
*/
ProblemNode* create_new_problem( Problem *p, Move *move )
{
    ProblemNode *newnode;
    Problem *newproblem;

    // allocate the new problem node
    newnode = (ProblemNode*) malloc( sizeof( ProblemNode ));

    // allocate the new problem data
    newproblem = (Problem*) malloc( sizeof( Problem ));
    newproblem->problem = newproblem;

    // copy the old problem information
    copy_problem( newnode->problem, p );

    // perform the given move on the old problem
    perform_move( newnode->problem, move );

    // adjust parameters within the new problem
    newnode->problem->whites_turn_flag = !newnode->problem->whites_turn_flag;
    newnode->problem->remaining_arch_levels--;
    find_attacked_squares( newproblem );

    return newnode;
void copy_problem( Problem *dest, Problem *src )
{
    copy_board( dest->board, src->board );
    copy_move( &dest->move, &src->move );
    dest->whites_turn_flag = src->whites_turn_flag;
    dest->solve_for_white_flag = src->solve_for_white_flag;
    dest->remaining_srch_levels = src->remaining_srch_levels;
}

void perform_move( Problem *p, Move *m )
{
    // delete the given piece if necessary
    if ( m->delete_loc.row != -1 ) {
        p->board[m->delete_loc.row][m->delete_loc.col].occupied = FALSE;
    }
    // handle the primary piece to be moved
    move_piece( p, m->src, m->dest );
    // handle the secondary piece to be moved
    move_piece( p, m->src2, m->dest2 );
    // create a new piece if necessary (only changed the piece type)
    if ( m->new_piece_loc.row != -1 ) {
        p->board[m->new_piece_loc.row][m->new_piece_loc.col].ptype = m->new_piece;
    }
    // save the performed move in the problem
    copy_move( &p->move, m );
}

void move_piece( Problem *p, PieceLoc src, PieceLoc dest )
{
    if ( src.row == -1 ) {
        return;
    }
    // copy the contents of the src square to the dest
    copy_board_square( p->board, src, dest );
    // label the src square as unoccupied
    p->board[src.row][src.col].occupied = FALSE;
    // label the piece as "moved"
    p->board[dest.row][dest.col].has_moved = TRUE;
}

/*
 * Destroy the given search list.
 */
void destroy_search_list( SearchList search_list[MAX_SEARCH_LEVEL],
                        int srch_level )
{
    ProblemNode *temp;
    while( search_list[srch_level].problem_start != NULL ) {
        temp = search_list[srch_level].problem_start;
        search_list[srch_level].problem_start = temp->next;
        free( temp );
        //delete temp;
    }
}

// Destroy the given solution list.
void destroy_solution_list( SolutionNode **slist )
{
    SolutionNode *temp=*slist;
    if ( slist == NULL ) return;
    while( temp != NULL ) {
        temp = (*slist)->next;
free( *slist );
*slist = temp;
}
)

9. References

None.