CS 740, Fall 1998

Assignment 6B: Parallel Branch-and-Bound for the Travelling Salesman Problem

Assigned: Thursday, Nov. 19
Due: 1:20 pm, Thursday, Dec. 3

General Information

In this assignment you will be solving the Travelling Salesman Problem (TSP) using the Branch-and-Bound technique. TSP is representative of combinatorial optimization problems. The Branch-and-Bound technique is an exhaustive evaluation technique that tries to make use of knowledge of the underlying problem to reduce the amount of computation.

You may work in groups of up to three for this assignment.

The Travelling Salesman Problem

The object of TSP is to find the shortest route for a traveling salesman so that the salesman visits every one of a set of cities exactly once and then returns home. This is a hard combinatorial optimization problem since for N cities there are at most (N-1)! possible routes (we assume that the cities are fully connected).

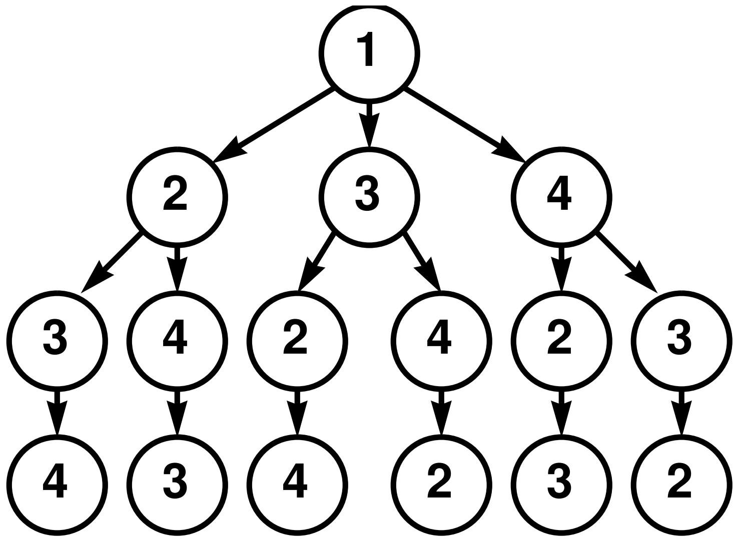
The input to the problem is given in the form of a matrix. An element of the matrix, *d[i][j]* gives the distance between city *i* and city *j*. The input to your program should be a file organized as follows:

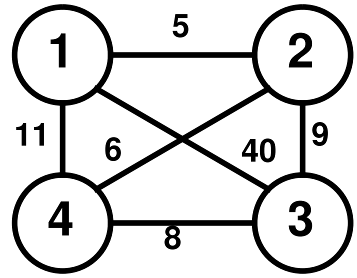
```
N
d[1][2]
d[1][3] d[2][3]
d[1][4] d[2][4] d[3][4]
     .
     .
     .
d[1][N] d[2][N] d[3][N] ... d[N-1][N]
```

where *N* is the number of cities, and *d[i][j]* is an integer giving the distance between cities *i* and *j*. The output from the program should be an ordered list of cities (numbers between 1 and N). Clearly, there are *N* equivalent permutations, so please use the one that starts with the first city.

Branch-and-Bound Solutions to Combinatorial Optimization Problems

First, consider a simple exhaustive evaluation as a way of solving the TSP. One way you might think about evaluating every possible route is to construct a tree that describes all of the possible routes from the first city, as shown on the next page for a four city example.





For this TSP, *d[1][2]* = 5, *d[1][3]* = 40, *d[1][4]* = 11, *d[2][3]* = 9, *d[2][4]* = 6, *d[3][4]* = 8. All of the possible routes can be found by traveling from the root to a different leaf node three times, once for each unique path and then back to the root. For example, by taking the middle branch from the root node and the right branch after that, the route 1->3->4->2->1 is produced, and has a distance of 40+8+6+5 = 59. There are six unique root to leaf paths in this tree. Each possible route is represented twise, once in each direction.

A simple exhaustive evaluation of TSP for this example would then be to follow the six root to leaf paths and determine the total distance for each path by adding up the distances indicated by each edge in the tree and then adding the distance back to the root. The route with the smallest distance is then chosen.

A better way to traverse each tree is to do it recursively. Here the summation of the earlier parts of each route is not repeated every time that route portion is reused in several routes.

The Branch-and-Bound approach uses this type of problem formulation, but with some added intelligence. It uses more knowledge of the problem to *prune* the tree as much as possible so that less evaluation is necessary. The basic approach works as follows:

1. Evaluate one route of the tree in its entirety, (say 1->2->3->4->1) and determine the distance of that path. Call this distance the current "bound" of the problem. The bound for this path in the above tree is 5+9+8+11=33.
2. Next, suppose that a second path is partially evaluated, say path 1->3, and the *partial distance*, 40, is already greater than the *bound*. If that is the case, then there is no need to complete the traversal of any part of the tree from there on, because all of those possible routes (in this case there are two) must have a distance greater than the bound. In this way the tree is *pruned* and therefore does not have to be entirely traversed.
3. Whenever any route is discovered that has a better distance than the current bound, then the bound is updated to this new value.

The Branch-and-Bound approach always remembers the best path it has found so far, and uses that to prevent search down parts of the tree that couldn't possibly produce better routes. You can see that for larger trees, this could result in the removal of many possible evaluations.

The Assignment

The assignment is to write a parallel Branch-and-Bound program for the TSP, using the ANL macros on the NCSA Origin 2000 machines. The objective is to obtain the best speedup possible. You should use the input format described above. You should have a command-line argument that says the number of processors to use. In addition, a command-line argument should also give the name of the input file.

Your report should include the following:

1. A brief (approximately one page) description of how your program works. Describe the general program flow and all significant data structures.
2. The solution to the problem given in *HOMEDIR*`/asst/asst6/distances`. This file contains a 12 city problem. (Hint: for debugging purposes, you may want to use some of the smaller input files included in the same directory.)
3. Execution time and speedup (both total and computation) for 1, 2, 4, 8, 16, 24, and 32 processors on an NCSA Origin 2000 machine. (If you can get even more processors, that is great.)
4. Discuss the results you expected and explain the reasons for any non-ideal behavior you observe. In particular, if you don't get perfect speedup, explain why. Is it *possible* to get better than perfect speedup? Give measurements to back up your explanations.
5. Include a printout of your program.

If the execution time for your program takes more than a few minutes on an Origin, double-check your program and algorithm. Once again, make sure your programs run on a uniprocessor before trying to run them on an Origin. Also, debug your programs using smaller numbers of cities (perhaps `HOMEDIR/asst/asst6/dist4`) and small numbers of processors before trying larger runs.