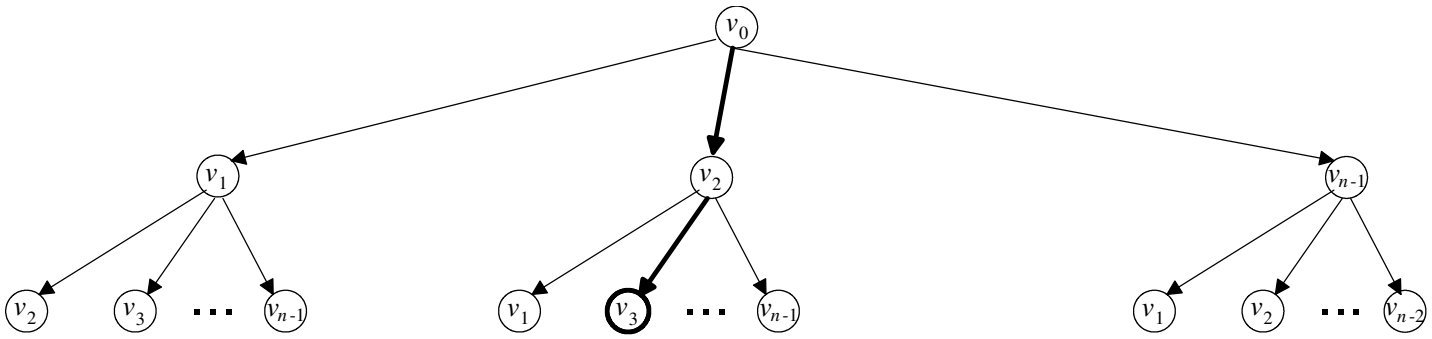
Program	Time	Comments
Sequential - DFS Backtracking Recursive, bound of minimum edges leaving remaining	158.553u 0.068s 2:38.63 99.9% (on Student)	Updates single state
Sequential - nonrecursive, bound of minimum edges leaving remaining tsp_iter2.c	2,127 seconds (35 min. and 27 sec.) (on Student)	Copies state on stack
pthread with dynamic allocation of work, bound of minimum edges leaving remaining pth_tsp_dyn.c with 8 threads	328 seconds (Wright 112 Lab on Intel i7)	
Sequential - DFS Backtracking Non-recursive tsp_nonrecursive.cpp	143.572u 0.008s 2:23.65 99.9% (on Student)	Updates single state

To parallelize the backtracking "tree" search using cuda, I made the following observations:

- updating a single state in much faster (2 minutes vs. 35 minutes)
- cuda kernels do not allow recursion so need to be nonrecursive
- hard for cuda threads to coordinate dynamically re-allocation of work, so statically assign work (lots of threads so if some get done early its not so bad)
- lots of threads so statically assign work via the host serially will limit speedup due to Amdahl's law - try to parallelize assigning work to threads initially (finally if static)

1. Consider the branch of the tree that is bold below.



a) What state information is needed by each approach?

```
Backtrack( recursionTreeNode p ) {
    treeNode c;
    for each child c of p do
            if promising(c) then
                    if c is a solution that's better than best then
                            best = c

                    else

                            Backtrack(c)

                    end if
            end if
    end for
} // end Backtrack
```

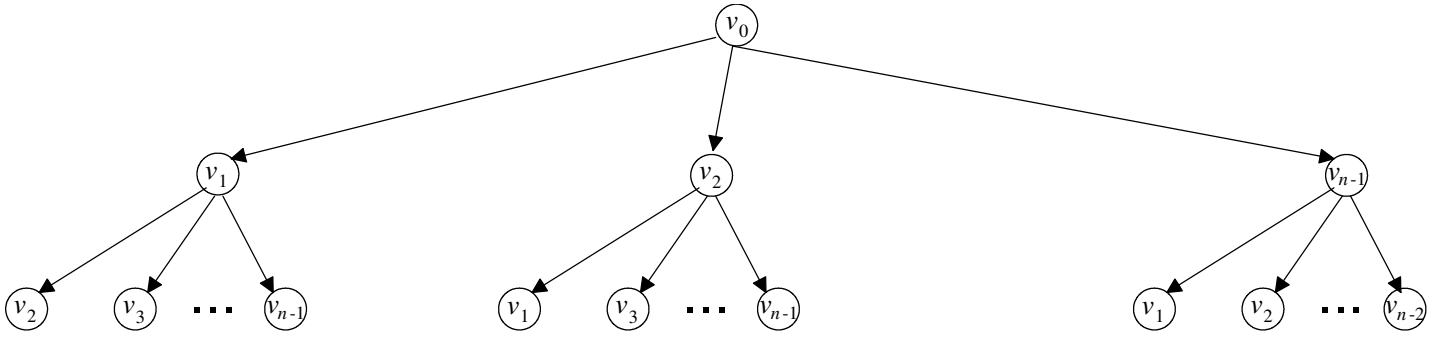
```
startState = Create partialTour with only v0
push(startState)
while stack is not empty do
   currentPartialTour = pop( )
   if currentPartialTour contains n cities then
      if currentPartialTour distance < bestTour distance then
          bestTour = currentPartialTour
    else
       for city not already on currentPartialTour do
           if  promising( currentPartialTour, city) then
                push( new lengthened partial tour with city added )
           end if
       end for
    end if
    free( currentPartialTour)
end while
```

2. Non-recursive DFS backtracking in tsp_nonrecursive.cpp:

```
void tsp() {
    int stack[MAX];// holds nextVertex of for-loop
    int level, pathLength, completePathLength, nextVertex, i;
    int popStack, possibleSibling, lastUpdateLevel, lastUpdateParent;

    level = 1;
    stack[0] = 1;
    stack[1] = 1;
    pathLength = 0;
    partialPath[0] = 0;

    while (TRUE) {
        nextVertex = stack[level];
        included[nextVertex] += 1;      // update global arrays to reflect child
        partialPath[level] = nextVertex;
        pathLength += E[partialPath[level-1]][nextVertex];
        if (nextVertex > 1) {  // remove siblings
            included[nextVertex-1] -= 1;      // update global arrays to reflect next child
            pathLength -= E[partialPath[level-1]][nextVertex-1];
        } // end if
        stack[level] = nextVertex + 1;

        if (nextVertex >= n) {
            level -= 1;
            popStack = TRUE;
            if (level <= 0)
                break;
            pathLength -= E[partialPath[level]][partialPath[level+1]];
            included[partialPath[level+1]] -= 1;    // change back to the parent's state
        } else {

            if (included[nextVertex] == 1) {  // only consider next vertices that are not already on the partial tour
                if (level == n-1) {          // for a complete tour see if it is the best so far
                    completePathLength = pathLength + E[nextVertex][0];
                    if (bestLength > completePathLength) {
                        bestLength = completePathLength;
                        for (i = 0; i < level; i++) {
                            bestPath[i] = partialPath[i];
                        } // end for(i...
                        bestPath[level] = nextVertex;
                        bestPath[level+1] = 0;
                    } // end (bestLength...
                } else if (promising(nextVertex, pathLength, level)) { // only check promising nodes
                    level = level + 1;
                    stack[level] = 1;
                    possibleSibling = FALSE;
                } // end if (level...
            } // end if (!included[nextVertex]...

        } // end if

    } // end while
} // end tsp
```

a) How does it keep track of the “backtracking” state?

3. If we wanted to split the work between two threads, how would we split the tree?



4. If we wanted to continue to split the work into fourths, how would we split the tree?