IELM 231. IT for Logistics and Manufacturing
Lab 5-6. Simple branch-and-bound searching

Objectives:
1. [from previous labs: connect to DB, and get data about a set of jobs to be scheduled]
2. Programming a simple Gantt chart for a schedule using html table.
3. Programming a branch-and-bound search method for a minimum cost, 1-m/c n-job schedule.

Introduction:
In the previous labs, you constructed a web-interface and a DB for an IT system to schedule jobs on one machine. In this lab, you will perform two tasks: Task 1. Using a simple SQL query, you can retrieve all the jobs that must be scheduled, ordered by due date. Then your CGI must create a simple Gantt chart displaying the EDD (earliest due date)-based schedule for these jobs, including the total lateness cost of this schedule. Details are given below.
Task 2. Using a simple SQL query, you will retrieve the data for all jobs. Then you must write a CGI program that uses a branch-and-bound search to find the minimum tardiness cost schedule for these jobs. Details are given below.

For both tasks, the CGI function must be written using VB 6.0. Please copy the CGI function of the previous lab, and modify the code to speed up your programming time.

Task 1. Making an EDD-based schedule:

Step 0.1. You already have a DB table, called JobsTable, as follows:
JobID (should be a unique integer),
ProcessTime (a 'double' denoting number of days to complete this task),
DueDate (a Date when the job must be completed), and
LateCost (a 'double' indicating $-penalty per day of tardiness in completion of the job.)

Step 0.2. The table already has the data of a few jobs that must be scheduled.

Step 1. Get the jobs data for EDD

In your CGI program, write and run an SQL query to get all the jobs in JobsTable. The rows of the output should be stored in the object *oRecordSet* in your CGI program.
[Hint: You can use the '***ORDER BY column_name***' clause in your SQL query to sort the output records by the values in ***column_name***.]

Step 2. Compute the total lateness cost

Create a VB object as follows in your CGI:

```
Public Type aJob
  job_id as integer
  proc_time as double
  due_date as date
  late_cost as double
End Type
```

Using a (global) array: Dim allJobs as aJob( ), and store all the jobs' data in this array (if your SQL query in Step 1 is correct, then the elements of this array are in EDD order). Note: you will have to ***ReDim allJobs*** to the correct size, depending on the number of jobs there are.

Write a function:

```
Public Function TardinessCost( ) as double

 'your function code goes here

End Function
```

This function will compute the lateness cost of all the jobs by sequentially looking at each job in the global array allJobs( ).

Step 3. Output a table showing the EDD schedule.

Hints:
- Since there is a single machine, the table will have only one row.

- The following example HTML code shows how to output a formatted table.

<table border=1 width=100%> <tr> <td width=25% bgcolor=#FF0000> Job_1 </td> <td width=75% bgcolor=#00FF00> Job_2 </td> </tr> </table>	make a table; table is 100% of web-page new row first cell, 25% of width of table, background, text is 'Job_1' Second cell, 75% of width of table, g background, text is 'Job_2' end of row end of table

- In your CGI program, compute the percent width of each cell depending on the Process time of the Job; to select color of each job, you can use alternately the shades Red and Green, or any other colors. Each color is defined simply by its R(ed), G(reen) and B(lue) value; each value is between 0-255, expressed in Hexadecimal -- i.e. any Hex number from 00 to FF.

Task 2. In this task, you can re-use some of the code from Task 1, but first you need to write a function for best first search using the same idea as in the lecture notes.

Step 1. Create a Node data structure:

```
Type Node
  job as integer
  prevJobs(20) as integer
  level as integer
  cost as double
End Type
```

Declare a global array of Open nodes: Dim openNodes( ) as Node. As your search goes on, you must ReDim openNodes( ) before adding more nodes to this array.

Hints:
- We shall assume the maximum number of jobs is 20.
- Level is an integer that stores how many jobs have already been assigned before this node. Thus, if level=0 then the array prevJobs ( ) is empty. If level=2, then the array prevJobs ( ) has two elements; suppose prevJobs(0)= 3 and prevJobs(1)= 5, and job= 1, then this node represents a partial solution where the last job scheduled is job 3, the 2nd last is job 5, and 3rd from last is job 1.

- Therefore we have no need to manage a list of CLOSED nodes: if the level in a node is equal to the total number of jobs, it is closed.

Step 2. Write a function: Public findCheapestOpenNode( ) as Integer

The function will go through the list of nodes in openNodes( ), and return the array index of the node which is currently the lowest cost open node.

Step 3. Write a function: Public Function goal(a_node as Node) as integer

First define a global variable: Public current_best as Node
Set the value of the cost of this node as a very large number.

The function checks if this node is a goal node [Hint: what is the relation between the number of jobs and the level of a node?];
if yes, then the function checks a global variable of type Node for its cost; if this goal node is cheaper than the cost of current_best, then replace current_best with a_node, and return 1.
if no, then return 0.

Step 4. Write a subroutine: Public Sub expand(this_node as Integer)

This subroutine looks at the node in openNodes(this_node), and creates a list of al its children;
For each child node, it computes the heuristic cost of this node;
It then updates the array openNodes( ) as follows:

The first child on this_node is used to replace openNodes(this_node); the remaining child nodes of this_node are added to the end of the array openNodes( ), after performing a ReDim openNodes(…).

Now you can complete to program the bf_schedule Subroutine using the above functions.
