Fall 2011 Name: _____

Algorithms Test 2

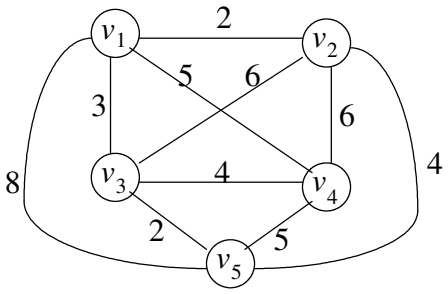
Question 1. (15 points) Prim's algorithm for determining the minimum-spanning tree (MST) of a undirected graph $G = (V, E)$ is an example of a *greedy algorithm*. The general idea of Prim's algorithm is:

Starts with any vertex, say v_1, in the initial, partial MST = (Y, F) with $Y = \{v_1\}$ and $F = \{\ \}$ (i.e., no edges yet).

Repeatedly grow the MST my adding one vertex at a time until it contains all the vertices in V

1. Choose a vertex not in the partial MST (from $V - Y$) that is closest to some vertex in the MST
2. Add the chosen vertex to the partial MST, i.e., add it the set Y
3. Add the edge that connected the chosen vertex to the MST to F

a) Trace the above algorithm on the below graph to find the MST. Show F (edges in the partial MST) by making the edge lines **bold and numbering the order (actually use letters: a, b, c, etc.) they are added to F.**



Question 2. (15 points) Both backtracking and branch-and-bound are techniques used to try to solve problems like the 0-1 knapsack problem whose best known algorithms are $\theta(2^n)$ or $\theta(n!)$.

a) How do backtracking and branch-and-bound attempt to solve reasonably large instances of these problems within a reasonable amount of time?

b) How do backtracking and branch-and-bound differ in their general philosophy of searching the state-space tree?

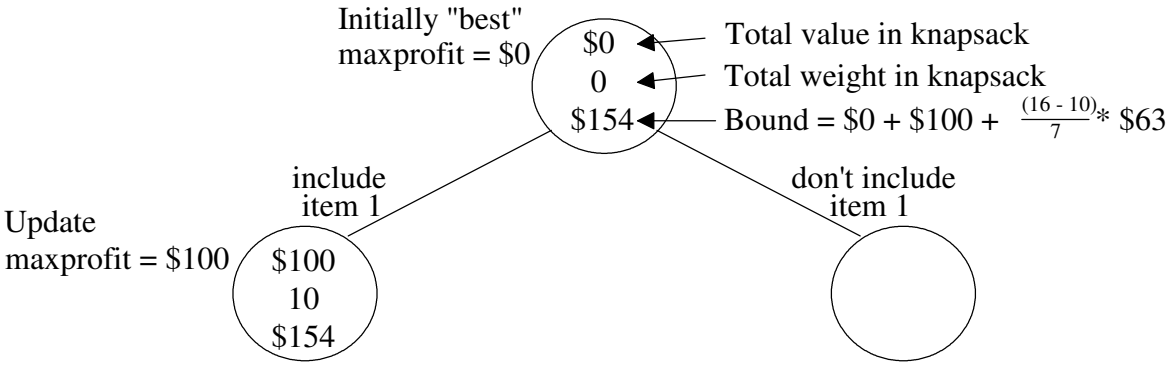
c) How do backtracking and branch-and-bound differ in their memory requirements?

Question 3. (30 points) Consider the following 0-1 Knapsack problem with five items and a knapsack weight limit of W=16 oz.

Item	Weight	Value	Value/Weight
1	10 oz.	$100	$10/oz.
2	7 oz.	$63	$9/oz.
3	8 oz.	$56	$7/oz.
4	4 oz.	$12	$3/oz.

a) Complete the **backtracking state-space tree**. Use a bound (e.g., best possible solution we could hope to achieve in the subtree) calculation of

bound of a node = value in knapsack + fractional knapsack problem on remaining items and weight.



b) **Without pruning** how many nodes would the complete state-space tree have contained?

c) If best-first search branch-and-bound was used, indicate the order in which nodes are expanded to their children.

Question 4. (40 points) The **CNF-Satisfiability Decision Problem** is to determine for a given CNF expression whether there exists some truth assignment that makes the CNF expression "TRUE."

Recall the terminology of CNF:

$x_1, x_2, ..., x_n$: Boolean variables
$\overline{x}_1, \overline{x}_2, ..., \overline{x}_n$: complement of the Boolean variables
$\overline{x}_1, \overline{x}_2, ..., \overline{x}_n, x_1, x_2, ..., x_n$: set of "literals"

A *clause* is the "or"ing ($\vee$) of a set of literals, e.g., $(\overline{x_1} \vee \overline{x_2} \vee x_4)$.

A Boolean expression is in *CNF (Conjunctive Normal Form)* if it is the conjunction/"anding" ($\wedge$) of one or more clauses, e.g., $(x_1 \vee x_2) \wedge x_3 \wedge (\overline{x_1} \vee \overline{x_2} \vee x_4)$.

For example, the CNF-SAT Boolean formula B = $(x_1 \vee x_2) \wedge x_3 \wedge (\overline{x_1} \vee \overline{x_2} \vee x_4)$ is "satisfiable" (decision problem would answer "Yes") since the true assignment of x_1=TRUE, x_2= FALSE, x_3=TRUE, x_4=TRUE makes B true.

You are to write an algorithm (English like steps are fine) to solve the **CNF-Satisfiability Problem.** For Boolean formulas that are satisfible, your algorithm should determine a true assignment to the Boolean variables to make the CNF expression TRUE. For Boolean formulas that are not satisfible, your algorithm should report "Not Satisfiable".

Select ONE of the following types of algorithms for your answer (any is acceptable):

- greedy algorithm - your algorithm does NOT need to always find a truth assignment even if the CNF expression is satisfiable. Include in your answer:
  - description of your greedy algorithm in English/pseudo-code
  - trace an example on a simple CNF expression
  - explain the worst-case run-time theta notation (i.e., $\Theta(\ )$) of your algorithm if the CNF expression has c clauses and n Boolean variables

- backtracking algorithm - your algorithm **must always** find a truth assignment if the CNF expression is satisfiable. Include in your answer:
  - description of your backtracking algorithm in English/pseudo-code by showing a picture of a state-space tree with information shown in each node
  - description of a promising function used to prune nodes (and their subtrees)
  - (partially) trace an example on a simple CNF expression by showning a (partial) state-space tree

- best-first search branch-and-bound algorithm - your algorithm **must always** find a truth assignment if the CNF expression is satisfiable. Include in your answer:
  - description of your best-first search algorithm in English/pseudo-code by showing a picture of a state-space tree with information shown in each node
  - description of a bound calculation used to guide expansion of tree nodes to their children
  - (partially) trace an example on a simple CNF expression by showning a (partial) state-space tree