## Lab 10: NP and NP Completeness

Consider the Dynamic Programming algorithm (that was covered in class) for 0-1 Knapsack problem. Fill in the table below: All problems have 10 items

|  |  |  |  |
| --- | --- | --- | --- |
| **Capacity W** | **# of bytes need to store W** | **Approximate Size of the Table** | **# of cells in the table** |
|  | 1 | W + 1 x 10 |  |
|  | 2 | W + 1 x 10 |  |
|  | 3 | W + 1 x 10 |  |
|  | 4 | W + 1 x 10 |  |
|  | 5 | W + 1 x 10 |  |

What is the computational complexity of the algorithm as a function of the size of the input W?

T(n, log C) = n \* 2log C

Variations on Knapsack

* O-1 Knapsack: A set of items (wi, vi), Capacity, maximize the value of items of weight ≤ Capacity. Each item can only be used once, a fraction of an item may not be used.
* Knapsack with repeats: A set of items (wi, vi), Capacity, maximize the value of items of weight ≤ Capacity. Each item has as many duplicates as desired, a fraction of an item may not be used.
* Continuous or Fractional Knapsack: A set of items (wi, vi), Capacity, maximize the value of items of weight ≤ Capacity. Each item can only be used once or a fraction of an item may be used.

Greedy approaches to knapsack: For each of the following greedy approaches to 0-1 Knapsack, give a counterexample where it does not find an optimal solution and that it is not a **c-approximation for any finite c**.

1. Heaviest item first:
2. Most valuable item first
3. Largest value/weight ration first

### Search and NP Completeness, Satisfiability

Preamble: The following problems are to increase your knowledge of satisfiability. Here we will consider a special case of satisfiability that is important in complexity theory of computation. One version of the problem is to determine a set of values that make a specific type of boolean expression true. This is considered a search problem since one can view it as searching over the space of all possible sets of values of the variables until you find a set that makes the expression true.

1. Consider the set of boolean expressions that consist of **or-clauses** (expressions with only boolean variables connected by ∨, and then these clauses connected by **“and”** ∧’s. Expressions of this types are said to be in **conjunctive normal form, CNF.**    
   A simple example with only one clause is (x ∨ y). If either x or y is true, then the expression would be true. It is an important problem (SAT) is to determine if such a CNF expression can evaluate to **true** for some assignment of true / false values to the boolean variables x, y, z, w … and their negations. Consider an expression in CNF form that contains a subexpression … (w) ∧ ( ∨ ) … . Any assignment where z is **true** shows that the entire expression will be false and thus z is **true** can be eliminated from consideration.
   1. Why?
   2. Consider: (x∨ y ∨z) ∧ ( x ∨ ) ∧ ( y ∨ ) ∧ ( z ∨ ). What must the values of x, y and z be? Explain why?

(x∨ y ∨z) ∧ ( x ∨ ) ∧ ( y ∨ ) ∧ ( z ∨ ).

x=T ( y ∨ ) ∧ ( z).

y=T ( z).

z=T T

1. Given (w ∨ x∨ y ∨z) ∧ (w ∨ ) ∧ ( x ∨ ) ∧ ( y ∨ ) ∧ ( ∨ ). Use backtracking (where each stage of the state space tree represents an assignment of a truth value to a boolean variable. **Assign the values to variables in order w, x, y, z**. As you assign values, new subproblems will be formed since some clauses will become true and thus can be safely ignored or it may be clear that a clause will be forced to be false in which case there is no need to explore any further down that subtree since you know the entire expression will be false for that subtree.

Submit a complete picture of your search tree obtained using backtracking and the variable ordering given above.

(w ∨ x∨ y ∨z) ∧ (w ∨ ) ∧ ( x ∨ ) ∧ ( y ∨ ) ∧ ( ∨ )

( x ∨ ) ∧ ( y ∨ ) ∧ ( )

( y ∨ ) ∧ ( )

( )

Solution

F

W

X

Y

Z

W = T, X = T, Y = T, Z = F

Definition (**Minimum** **Vertex Cover Problem**): A **vertex cover of size m** for a graph G = (V,E) is a subset V’ ⊆ V such that |V’| = m and, for each edge (u,v) ∈ E at least one of u and v belongs to V’ . Determine, for a given graph G = (V,E) a minimum size **vertex cover**

1. Give pseudo code for a 2-approximation algorithm for the vertex cover problem
2. Prove that your algorithm is a 2-approximation. (Hint: There is an algorithm that finds a maximal matching for a graph G in polynomial time.) Recall:

* Given a graph G = (V,E), a **matching** M in G is a set of pairwise non-adjacent edges, none of which are loops; that is, no two edges share a common vertex. A vertex is **matched** if it is an endpoint of one of the edges in the matching. Otherwise the vertex is unmatched.
* A **maximal matching** is a matching M of a graph G with the property that if any edge not in M is added to M, it is no longer a matching, that is, M is **maximal** if it is not a subset of any other matching in graph G. In other words, a matching M of a graph G is maximal if every edge in G has a non-empty intersection with at least one edge in M. The following figure shows examples of maximal matchings (red) in three graphs.
* A **maximum matching** (also known as maximum-cardinality matching) is a matching that contains the largest possible number of edges. Every maximum matching is maximal, but not every maximal matching is a maximum matching.

**Definition**: Given a Graph G = (V, E), define the complement graph of G, , to be = (V, ) where is the complement set of edges. That is (v,w) is in if and only if (v,w) ∉ E .

1. **Prove the following**

**Theorem**: Given G, the complement graph of G, can be constructed in polynomial time.

**Definition**: **Vertex Cover Decision Problem** is to determine for a given graph G = (V,E) and a positive integer m ≤ |V |, whether there is a vertex cover of size m or less for G (A **vertex cover** of size m for a graph G = (V,E) is a subset V’ ⊆ V such that |V’| = m and, for each edge (u,v) ∈ E at least one of u and v belongs to V’.)

**Definition:** **Clique Decision Problem** is to determine for a given graph G = (V,E) and a positive integer m ≤ |V |, whether G contains a clique of size m or more. A **clique of size k** for a graph G = (V,E) is a subset V’ ⊆ V such that |V’| = k and for all u, v ∈ V’, the vertices u and v are adjacent. That is the subgraph determined by restricting G to V’ is complete.

1. **Prove the following**

**Theorem**: C is a vertex cover in a graph G= (V, E) if and only if the set of vertices V-C is a clique in the complement graph of G.

1. **Prove: C a vertex cover implies that V-C is a clique**
2. **Prove: V-C is a clique implies C a vertex cover**
3. **Prove the following**

**Theorem:** Min Vertex cover and Max Clique are polynomial reducible to each other. (Thus if you can solve one problem in polynomial time you can solve the other problem in polynomial time.)