CS 373: Combinatorial Algorithms, Fall 2002

http://www-courses.cs.uiuc.edu/~cs373

Homework 6 (Do not hand in!)

Name:		
Net ID:	Alias:	U ¾ 1

Name:		
Net ID:	Alias:	U ¾ 1

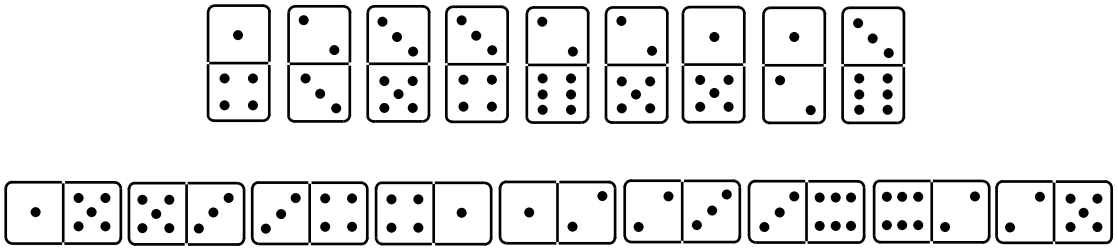
Name:		
Net ID:	Alias:	U ¾ 1

Neatly print your name(s), NetID(s), and the alias(es) you used for Homework 0 in the boxes above. Please also tell us whether you are an undergraduate, 3/4-unit grad student, or 1-unit grad student by circling U, ¾, or 1, respectively. Staple this sheet to the top of your homework.

Required Problems

1. (10 points) Prove that SAT is still a NP-complete problem even under the following constraints: each variable must show up once as a positive literal and once or twice as a negative literal in the whole expression. For instance, $(A \vee \bar{B}) \wedge (\bar{A} \vee C \vee D) \wedge (\bar{A} \vee B \vee \bar{C} \vee \bar{D})$ satisfies the constraints, while $(A \vee \bar{B}) \wedge (\bar{A} \vee C \vee D) \wedge (A \vee B \vee \bar{C} \vee \bar{D})$ does not, because positive literal A appears twice.

2. (10 points) A domino is 2×1 rectanble divided into two squares, with a certain number of pips(dots) in each square. In most domino games, the players lay down dominos at either end of a single chain. Adjacent dominos in the chain must have matching numbers. (See the figure below.)
   Describe and analyze an efficient algorithm, or prove that it is NP-complete, to determine wheter a given set of n dominos can be lined up in a single chain. For example, for the sets of dominos shown below, the correct output is TRUE.



Top: A set of nine dominos
Bottom:The entire set lined up in a single chain

3. (10 points) Prove that the following 2 problems are NP-complete. Given an undirected Graph $G = (V, E)$, a subset of vertices $V' \subseteq V$, and a positive integer k:

   (a) determine whether there is a spanning tree T of G whose leaves are the same as V'.

   (b) determine whether there is a spanning tree T of G whose degree of vertices are all less than k.

4. (10 points) An optimized version of Knapsack problem is defined as follows. Given a finite set of elements U where each element of the set $u \in U$ has its own size $s(u) > 0$ and the value $v(u) > 0$, maximize $A(U') = \sum_{u \in U'} v(u)$ under the condition $\sum_{u \in U'} s(u) \leq B$ and $U' \subseteq U$. This problem is NP-hard. Consider the following polynomial time approximation algorithm. Determine the worst case approximation ratio $R(U) = \max_{U} Opt(U)/Approx(U)$ and prove it.

```
ApproximationAlgorithm:
    A1 ← Greedy()
    A2 ← SingleElement()
    return max(A1, A2)
```

```
Greedy:
    Put all the elements u∈U into an array A[i]
    Sort A[i] by v(u)/s(u) in a decreasing order
    S ← 0
    V ← 0
    for i ← 0 to NumOfElements
        if (S + s(u[i]) > B)
            break
        S ← S + s(u[i])
        V ← V + v(u[i])
    return V
```

```
SingleElement:
    Put all the elements u∈U into an array A[i]
    V ← 0
    for i ← 0 to NumOfElements
        if (s(u[i]) ≤ B & V < v(u[i]))
            V ← v(u[i])
    return V
```

5. (10 points) The recursion fairy's distant cousin, the reduction genie, shows up one day with a magical gift for you: a box that determines in constant time whether or not a graph is 3-colorable.(A graph is 3-colorable if you can color each of the vertices red, green, or blue, so that every edge has do different colors.) The magic box does not tell you how to color the graph, just wheter or not it can be done. Devise and analyze an algorithm to 3-color any graph in **polynomial time** using the magic box.

6. (10 points) The following is an NP-hard version of PARTITION problem.

PARTITION(NP-HARD):

Given a set of n positive integers $S = \{a_i | i = 0 \ldots n-1\}$,

minimize $\max\left(\sum_{a_i \in T} a_i, \sum_{a_i \in S-T} a_i\right)$

where T is a subset of S.

A polynomial time approximation algorithm is given in what follows. Determine the worst case approximation ratio $\min_{S} Approx(S)/Opt(S)$ and prove it.

```
ApproximationAlgorithm:
    Sort S in an increasing order
    s1 ← 0
    s2 ← 0
    for i ← 0 to n
        if s1 ≤ s2
            s1 ← s1 + a_i
        else
            s2 ← s2 + a_i
    result ← max(s1, s2)
```

Practice Problems

1. Construct a linear time algorithm for 2 SAT problem.

2. Assume that $P \neq NP$. Prove that there is no polynomial time approximation algorithm for an optimized version of Knapsack problem, which outputs $A(I)$ s.t. $|Opt(I) - A(I)| \leq K$ for any instance I, where K is a constant.

3. Your friend Toidi is planning to hold a party for the coming Christmas. He wants to take a picture of all the participants including himself, but he is quite **shy** and thus cannot take a picture of a person whom he does not know very well. Since he has only **shy** friends, every participant coming to the party is also **shy**. After a long struggle of thought he came up with a seemingly good idea:

   - At the beginning, he has a camera.
   - A person, holding a camera, is able to take a picture of another participant whom the person knows very well, and pass a camera to that participant.
   - Since he does not want to waste films, everyone has to be taken a picture exactly once.

   Although there can be some people whom he does not know very well, he knows completely who knows whom well. Therefore, in theory, given a list of all the participants, he can determine if it is possible to take all the pictures using this idea. Since it takes only linear time to take all the pictures if he is brave enough (say "Say cheese!" N times, where N is the number of people), as a student taking CS373, you are highly expected to give him an advice:

   - show him an efficient algorithm to determine if it is possible to take pictures of all the participants using his idea, given a list of people coming to the party.
   - or prove that his idea is essentially facing a NP-complete problem, make him give up his idea, and give him an efficient algorithm to practice saying "Say cheese!":
     e.g., `for i ← 0 to N`
`    Make him say "Say cheese!" 2^i times` oops, it takes exponential time...

4. Show, given a set of numbers, that you can decide wheter it has a subset of size 3 that adds to zero in polynomial time.

5. Given a CNF-normalized form that has at most one negative literal in each clause, construct an efficient algorithm to solve the satisfiability problem for these clauses. For instance,

$$(A \vee B \vee \bar{C}) \wedge (B \vee \bar{A}),$$
$$(A \vee \bar{B} \vee C) \wedge (B \vee \bar{A} \vee D) \wedge (A \vee D),$$
$$(\bar{A} \vee B) \wedge (B \vee \bar{A} \vee C) \wedge (C \vee D)$$

satisfy the condition, while

$$(\bar{A} \vee B \vee \bar{C}) \wedge (B \vee \bar{A}),$$
$$(A \vee \bar{B} \vee C) \wedge (B \vee \bar{A} \vee \bar{D}) \wedge (A \vee D),$$
$$(\bar{A} \vee B) \wedge (B \vee \bar{A} \vee C) \wedge (\bar{C} \vee \bar{D})$$

do not.

6. The ExactCoverByThrees problem is defined as follows: given a finite set X and a collection C of 3-element subsets of X, does C contain an exact cover for X, that is, a sub-collection $C' \subseteq C$ where every element of X occurs in exactly one member of C'? Given that ExactCoverByThrees is NP-complete, show that the similar problem ExactCoverByFours is also NP-complete.

7. The *LongestSimpleCycle* problem is the problem of finding a simple cycle of maximum length in a graph. Convert this to a formal definition of a decision problem and show that it is NP-complete.