上 海 交 通 大 学 试 卷（A卷）

（ 2014 至 2015 学年 第 2 学期 ）

学　　号：　　姓名：

课程名称：　算法设计与分析　　成绩：

1. (10 points) Write the dual to the following linear program.

$$\text{Max } 6x - 4z - 3$$
$$3x - y \leq 1$$
$$4y - z \leq 2$$
$$x, y, z \geq 0$$

Is the solution $(x, y, z) = (1/2, 1/2, 0)$ optimal? Write the dual program of the given linear program and find out its optimal solution.

Sol: Dual program:

Min x+2y-3

subject to 3x ≥ 6

-x+4y ≥ 0

-y ≥ -4

x, y ≥ 0

(x, y, z) = (1/2,1/2,0) is optimal.

我承诺，我将严格遵守考试纪律。

承诺人：_____

题号									
得分									
批阅人(流水阅卷教师签名处)									

2. (10 points) Give an $O(n \cdot t)$ algorithm for the following task.
Input: A list of n positive integers $a_1, a_2, \ldots a_n$, and a positive integer t.
Question: Does some subset of the a_i's add up to t? (You can use each a_i at most once)

Sol:

Define $f(i,j) = \begin{cases} true, if\ some\ subset\ of\ a_1, a_2, \ldots a_i\ add\ up\ to\ j \\ false, otherwise \end{cases}$

Then $f(i,j) \begin{cases} true,\ if\ \ i = 0\ \ and\ \ j = 0 \\ true,\ if\ \ f(i-1,j) = true \\ true,\ if\ \ j \geq a_i\ \ and\ \ f(i-1, j-a_i) = true \\ flase, otherwise \end{cases}$

A natural dynamic programming algorithm:

Initialize $f(0,0) = \text{true}$ $f(0,i) = false (1 \leq i \leq t)$

For $i = 1$ to n

For $j = 0$ to t

If $f(i-1,j) = \text{true}$ or $(j \geq a_i\ \ and\ \ f(i-1, j-a_i) = \text{true})$

Then $f(i,j) = \text{true}$

Else $f(i,j) = \text{false}$

Return $f(n,t)$

3. (10 points) In a **facility location problem**, there is a set of facilities and a set of cities, and our job is to choose a subset of facilities to open, and to connect every city to some one of the open facilities. There is a nonnegative cost f_j for opening facility *j*, and a nonnegative connection cost $c_{i,j}$ for connecting city i to facility j. Given these as input, we look for a solution that minimizes the total cost. Formulate this facility location problem as an integer programming problem.

Sol:

Minimize $\sum f_j x_j + \sum c_{i,j} x_j y_{i,j}$ subject to

$\sum x_j y_{i,j} \geq 1$ for any i

$x_i \geq 0$ for any i

$y_{i,j} \geq 0$ for any i, j

x_i denotes whether facility i opens,

$y_{i,j}$denotes whether city i to facility j

4. (15 points) Given a reduction from the **Clique Problem** to the following problem, which you also need to show to be a search problem.

Input a undirected graph *G* and a positive integer k.

Output a Clique of size k as well as an Independent Set of size k, provided both exist.

Sol: 1. Since given an answer of the problem, we can check whether the given clique and independent set are correct in polynomial time, so it's a search problem.

2. We now give a reduction from the Clique Problem to this problem:

Clique problem: *Given a undirected graph G and a positive integer k, ask whether the graph G has a clique of size k.*

The reduction is very simple, we just construct the graph $G = (V \bigcup V', E)$, where $|V'| = k$. Then if G' has a clique of size k > 1, all the nodes are in the set V. So G has a clique of size k if and only if G' has both a clique of size k and an independent set of size k. Since the reduction is in polynomial time, we have finished the reduction.

5. (15 points) You are given an undirected graph. The problem is to remove a minimum number of edges such that the residual graph contains no triangle. i.e., there is no three vertices *a, b, c* such that edges *(a, b),(b, c),(c, a)* are all in the residual graph. Give a factor 3 approximation algorithm that runs in polynomial time.

Sol:

Executive summary: The basic idea is to repeatedly find triangles in the graph G, and remove all three edges from the triangle. Since any optimal algorithm has to remove at least one edge from the triangle, we conclude that it is a 3-approximation algorithm.

Algorithm description: First, we need a sub-routine that, given a graph G = (V, E), finds a triangle. Assume that the graph is stored as an adjacency matrix, and consider the simple algorithm that examines every combination of three nodes (u, v, w) and checks whether there are edges connecting them. In this way, it takes $O(n^3)$ time to find a triangle (or report that there are no triangles). Certainly, further optimization is possible.

Second, the algorithm proceeds as follows: while there are any triangles left in the graph, execute the following steps:

• Let (u, v, w) be a triangle in G.

• Remove edges (u, v), (v, w), and (u, w) from G.

Explain why it works: First, it is easy to see that when the algorithm completes, there are no triangles. It remains to show that the algorithm is a 3-approximation algorithm. Define a *triangle-set* of G to be a set of disjoint triangles in G, i.e., a set of triangles such that no two triangles share an edge. (Triangles may share a node, however.) Notice that the algorithm produces a triangle-set, since after each iteration of the loop, it removes the edges from the previous triangle from the graph. Moreover, if the algorithm produces triangle set S, then the number of edges removed by the algorithm is exactly 3|S|.

Let OPT be the optimal set of edges to remove from the graph G to ensure that it is triangle-free. Notice that for any triangle-set S, we can conclude that |OP T| ≥ |S|, since OPT has to remove at least one edge in each triangle in S. Putting this together with the previous claim, we conclude that the number of edges removed by the algorithm is 3|S| ≤ 3|OPT|, and hence the algorithm is a 3-approximation of optimal.

Running time: The running time of the algorithm is at most O(n3m): each iteration of triangle finding takes time O(n3), and there are at most m such iterations (since each iteration removes 3 edges).

6. (15 points) A **Minimum Makespan Scheduling** problem is as follows:

Input processing times for n jobs, $p_1, p_2, \ldots, p_n$, and an integer m.

Output an assignment of the jobs to m identical machines so that the completion time is minimized.

Design an approximation algorithm by a greedy approach on the problem, with the approximation factor 2. And also give a tight example to show the approximation guarantee.

Sol:

Algorithm:

1. Order the jobs arbitrarily.

2. Schedule jobs on machines in this order, scheduling the next job on the machine that has been assigned the least amount of work so far.

A tight example for this algorithm is provided by a sequence of m^2 jobs with unit processing time, followed by a single job of length m. The schedule obtained by the algorithm has a makespan of $2m$, while $OPT = m + 1$.

7. (10 points) The **Maximum Cut** problem is defined as follows: Given an undirected graph $G = (V, E)$ along with a nonnegative weight $w_{ij} \geq 0$ for each $(i, j) \in E$. The goal is to partition the vertex set into two parts, U and $W = V - U$, so as to maximize the weight of the edges whose two endpoints are in different parts. Give a 2-approximation randomized algorithm for maximum cut problem.

Sol: The algorithm is very easy, we just need place each vertex $v \in V$ into U independently with probability $\frac{1}{2}$.

Algorithm Analysis: Consider a random variable X_{ij} that is 1 if the edge (i, j) is in the cut, and 0 o.w. . Let Z be the random variable equal to the total weight of edges in the cut, so that $Z = \sum_{(i,j)\in E} w_{ij} X_{ij}$. Let OPT denote that the optimal value of maximum cut instance. Then, as before by linearity of expectation and the definition of expectation of 0-1 random variable, we get that:

$$E[Z] = \sum_{(i,j)\in E} w_{ij} E[X_{ij}] = \sum_{(i,j)\in E} w_{ij} \Pr[Edge(i,j) \text{ in the cut}]$$

In this case, the probability that a specific edge (i,j) is in the cut is easy to calculate: since the two endpoints are placed in the sets independently, they are in different sets with probability equal to. Hence,

$$E[Z] = \frac{1}{2} \sum_{(i,j)\in E} w_{ij} \geq \frac{1}{2} OPT$$

8. (15 points) Suppose in a given network, all edges are undirected (or think of every edge as bi-directional with the same capacity), and the length of the longest simple path from the source s to sink t is at most *L*. Show that the running time of **Edmond-Karp algorithm** on such networks can be improved to be $O(L \cdot |E|^2)$.

Sol:

Algorithm 2 EDMONDSKARP(G)

1: $f \leftarrow 0$; $G_f \leftarrow G$
2: **while** G_f contains an $s-t$ path P **do**
3: Let P be an $s-t$ path in G_f with the minimum number of edges.
4: Augment f using P.
5: Update G_f
6: **end while**
7: **return** f

The key ideas are:

- In each iteration in the body of while loop (line 3-5) the running time is $O(E)$
- The iteration times is at most $O(LE)$
  - In each iteration, at least one edge on the augmenting path must be *critical* (An edge is critical if the residual capacity of the augmenting path p equals the residual capacity of the edge (u, v), that is, $c_f(p) = c_f(u, v)$).
  - From the time (u, v) becomes critical to the time when it next becomes critical, the distance of u from the source increases by at least 2. Since the distance of u from the source is at most L, an edge can become critical at most O(L) times. (Need elaboration with mathematical proofs)
  - There are $O(E)$ pair of vertices that can have an edge between them in a residual network
  - Each augmenting path has at least one critical edge

So, the total time is $O(L \cdot |E|^2)$.

For a more detailed proof, see the book **Introduction to Algorithms**, from page 727 to page 730. http://basics.sjtu.edu.cn/~liguoqiang/teaching/Galgo17/books/IntroductiontoAlgorithms3rd.pdf