Fall '15 CS 5800 Algorithms

Problem Set 5

Surekha Jadhwani, Akash Singh, and Ayush K. Singh

1 Part A

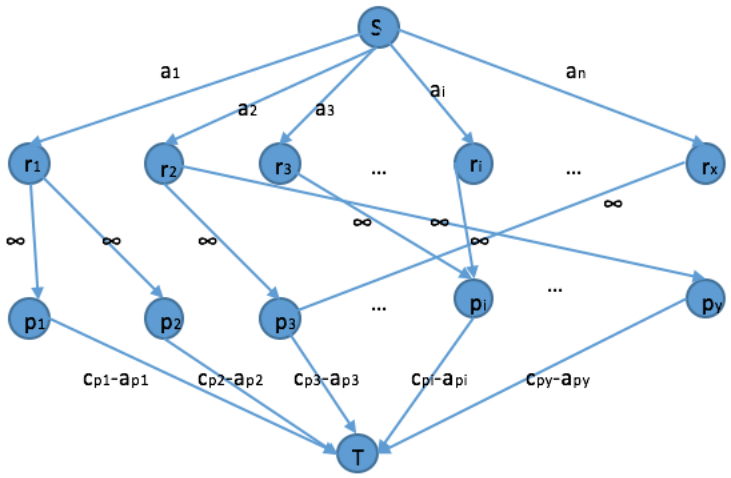
Answers

1. As you probably knows, the State of California has suffered from droughts in the last few years. The lack of rain caused the **water crisis** in California. In order to handle the water crisis, the government announced a renovation project to upgrade Californias reservoirs.
   This coming summer a subset of reservoirs are scheduled to be renovated. In the first stage it is required to deplete these reservoirs - all these reservoirs should be empty at the beginning of the renovation project. The government would like to save water (water of these reservoirs undergoing renovation) and hence decided to pump the water from these reservoirs (the ones being renovated this summer) to the other reservoirs (the ones not being renovated this summer).
   The pipe network between the reservoirs is not complete. There are pipes from each reservoir to a subset of the other reservoirs. It is known that there is enough overall capacity in the reservoirs not being to renovated to contain all the water in all the reservoirs. However, the project manager would like to save energy, and hence would like to (only) pump water from reservoirs undergoing renovation directly to other reservoirs (which are not scheduled for renovation).
   Let $R = \{1, 2, , n\}$ denotes the set of reservoirs and let $\widehat{R} \subseteq R$ denote the subset of reservoirs that are scheduled for renovation.
   For every reservoir $r_i \in R$, let c_i be the maximum capacity of reservoir r_i(the max amount of water that r_i can contain); and a_i the current amount of water that is in r_i. We have that $0 \leq a_i \leq c_i$. For every reservoir $r_i \in \widehat{R}$ which is scheduled for renovation, let $P_i \subseteq R \setminus \widehat{R}$ be the set of reservoirs that are not scheduled for renovation and are directly connected to r_i (that is, water of reservoir r_i can be directly pumped into reservoirs of P_i).

   - Give an efficient algorithm to decide whether all reservoirs of $\widehat{R}$ can be emptied by directly pumping their water into reservoirs of $R \setminus \widehat{R}$.

   Solution: Algorithm Idea: From the given inputs, we can construct a graph by taking reservoirs as the vertexes of the graph and adding a edge between two vertexes if first can be emptied into second. We can then convert this graph to flow network and then use max-flow algorithm to find whether all reservoirs of $\widehat{R}$ can be emptied directly or not to remaining reservoirs.
   Detailed Steps: The given problem can be reduced to max-flow/min-cut problem.
   In order to construct the graph, let us consider the set of reservoirs $R = \{1, 2, , n\}$ to

be the vertexes of the graph.
We can convert the above graph into directed graph by partitioning the vertexes into two sets($\widehat{R}$ and $R \setminus \widehat{R}$).
For each $r_i \in \widehat{R}$ which is scheduled for renovation, we will add directed edges from that reservoir to the reservoirs that are not scheduled for renovation and are directly connected to r_i (i.e., set P_i).
We can get all of the above information from the given input.
Now, in order to apply max-flow algorithm, we will convert this graph to a flow network by adding a super source vertex S which will be connected to all $r_i \in \widehat{R}$.
And a super sink node T which will be connected to all reservoirs(vertexes) in $R \setminus \widehat{R}$.
Now, we will add the capacities to the edges as follows:
From source S to all $r_i \in \widehat{R}$, the edge capacity would be equal to the amount of water stored in that reservoir, i.e., a_i. This is because the maximum amount of water that can be transferred is equal to the amount of water stored in the reservoir currently.
From all reservoirs in $R \setminus \widehat{R}$ to the sink T, the edge capacity would be equal to the amount of water that can be further stored in that reservoir, i.e., the total capacity of reservoir - the amount of water already stored in it $(c_i - a_i)$.
Also, the capacities for the edges connecting $r_i \in \widehat{R}$ and all reservoirs in $R \setminus \widehat{R}$ vertexes is ∞.
Now, this is converted to a flow network where the edge capacities have integer values as shown below:



Where:

S: represents super source

T: represents super sink

r_i: set of reservoirs undergoing renovation

p_i: remaining reservoirs

a_i: amount of water stored currently in r_i

c_{pi}: total capacity of p_i reservoir

a_{pi}: amount of water stored currently in p_i

Hence, to find the max-flow, we can use Edmonds-Karp network flow algorithm.
If the result is equivalent to $\sum a_i \ \forall r_i \in \widehat{R}$, then all reservoirs of $\widehat{R}$ can be emptied by directly pumping their water into reservoirs of $R \setminus \widehat{R}$, otherwise it cannot be done.

Complexity: We have total n reservoirs. Let $n-k$ be the number of reservoirs undergoing renovation, i.e., $|\widehat{R}| = n-k$. Therefore, the number of reservoirs not undergoing renovation is k, i.e., $|R \setminus \widehat{R}| = k$.
For the graph $G = (V, E)$, the number of vertexes is equal to all reservoirs, source and sink, i.e., $|V| = (n+2)$. For a sufficiently large value of n, we can approximate $n+2$ to n.
Also, the number of edges is equal to sum of edges from source to $(n-k)$ nodes and k nodes to sink and maximum possible edges between $(n-k)$ and k vertexes, i.e., $(n-k)k$. Thus, $|E| = n-k+k+(n-k)k = n+(n-k)k$
Hence, in order to construct the graph $G = (V, E)$, the worst-case complexity is $(O(n^2)$.
Since, we have used Edmonds-Karp algorithm to calculate max-flow, the worst case complexity is $O(|V||E|^2)$.
By substituting the above values of $|V|$ and $|E|$, we get the complexity as $O(n * [n+$

$(n-k)k]^2)$
Let $(n-k) = m$. Substituting and expanding, we get the complexity as $O(n^3+2n^2mk+nm^2k^2) \approx O(n^3k^2)$.
Hence, the worst case complexity is $O(n^3k^2)$.

2. Questions

(a) Assuming

$$\begin{aligned} \text{Min } & 5x + 3y - 4z \\ \text{s.t. } & 2y - 4z = 7 \\ & 4x - 2y \geq 5 \\ & 3x + 5z \leq -3 \\ & x \geq 0 \\ & y \geq 0 \end{aligned}$$

Rewrite the above linear program in

i. Canonical Form.
Solution: The canonical form(matrix-vector notation) is represented as:

$$\begin{aligned} \text{Max } & c^\intercal x \\ & Ax \leq b \\ & x \geq 0 \end{aligned}$$

Converting the given problem to canonical form using following steps:

A. Maximizing the Objective function:
Multiplying the objective function by -1, we get,

$$\text{Max } \; -5x - 3y + 4z$$

B. Changing the equality constraints to less than inequality constraint:

$$\begin{aligned} 2y - 4z &\leq 7 \\ 2y - 4z &\geq 7 \end{aligned}$$

C. Changing the greater than inequality constraint to less than inequality constraint:
Multiplying both sides by -1, we get

$$\begin{aligned} -2y + 4z &\leq -7 \\ -4x + 2y &\leq -5 \end{aligned}$$

D. Restricting free variable z to non-negative value:
Introducing two new variables z^+ and z^- such that $z^+, z^- \geq 0$ and $z =$

$z^+ - z^-$, we get the final converted problem as:

$$\begin{aligned}
\text{Max } & -5x - 3y + 4z^+ - 4z^- \\
\text{s.t. } & 2y - 4z^+ + 4z^- \leq 7 \\
& -2y + 4z^+ - 4z^- \leq -7 \\
& -4x + 2y \leq -5 \\
& 3x + 5z^+ - 5z^- \leq -3 \\
& x \geq 0 \\
& y \geq 0 \\
& z^+ \geq 0 \\
& z^- \geq 0
\end{aligned}$$

E. Creating matrices for canonical form as:

$$c = \begin{bmatrix} -5 \\ -3 \\ 4 \\ -4 \end{bmatrix}$$

Therefore, $c^\intercal = \begin{bmatrix} -5 & -3 & 4 & -4 \end{bmatrix}$

$$x = \begin{bmatrix} x \\ y \\ z^+ \\ z^- \end{bmatrix}$$

$$A = \begin{bmatrix} 0 & 2 & -4 & 4 \\ 0 & -2 & 4 & -4 \\ -4 & 2 & 0 & 0 \\ 3 & 0 & 5 & -5 \end{bmatrix}$$

$$b = \begin{bmatrix} 7 \\ -7 \\ -5 \\ -3 \end{bmatrix}$$

F. Hence, the final canonical form is:

$$\text{Max } \begin{bmatrix} -5 & -3 & 4 & -4 \end{bmatrix} \begin{bmatrix} x \\ y \\ z^+ \\ z^- \end{bmatrix}$$

$$\text{s.t. } \begin{bmatrix} 0 & 2 & -4 & 4 \\ 0 & -2 & 4 & -4 \\ -4 & 2 & 0 & 0 \\ 3 & 0 & 5 & -5 \end{bmatrix} \begin{bmatrix} x \\ y \\ z^+ \\ z^- \end{bmatrix} \leq \begin{bmatrix} 7 \\ -7 \\ -5 \\ -3 \end{bmatrix}$$

$$\begin{bmatrix} x \\ y \\ z^+ \\ z^- \end{bmatrix} \geq 0$$

ii. Standard Form.

Solution: The standard form(slack form) has following properties:

- Minimize objective function.
- All constraints are in the form of equations(equality) other than restricting variables.
- Variables are non-negative

Converting the given problem to standard form using following steps:

A. No change in the Objective function(already minimization).

B. Changing the greater than inequality constraint to equality constraint: Adding surplus variable s_1, we get

$$4x - 2y + s_1 = 5$$

C. Changing the less than inequality constraints to equality constraint: Adding slack variable s_2, we get,

$$3x + 5z - s_2 \leq -3$$

D. Restricting free variable z to non-negative value:
Introducing two new variables z^+ and z^- such that $z^+, z^- \geq 0$ and $z = z^+ - z^-$, we get the final converted problem into standard form as:

$$\begin{aligned} \text{Min } & 5x + 3y - 4z^+ + 4z^- \\ \text{s.t. } & 2y - 4z^+ + 4z^- = 7 \\ & 4x - 2y + s_1 = 5 \\ & 3x + 5z^+ - 5z^- = -3 \\ & x \geq 0 \\ & y \geq 0 \\ & z^+ \geq 0 \\ & z^- \geq 0 \end{aligned}$$

(b) Give an example each of an LP that is

i. Feasible and bounded.
Solution: Example of a feasible and bounded LP is:

$$\begin{aligned} \text{Max } & 5x + 4y \\ \textbf{s.t. } & x \leq 1000 \\ & y \leq 500 \\ & x + y \leq 700 \\ & x, y \geq 0 \end{aligned}$$

Here, the constraints for x and y is such that we can find an optimal solution for the objective function. Hence, above LP is feasible and bounded.

ii. Feasible but unbounded.
Solution: Example of a feasible but unbounded LP is:

$$\begin{aligned} \text{Max } & x + y \\ \textbf{s.t. } & x, y \geq 0 \end{aligned}$$

Here, the constraint for x and y is so loose, that it is impossible to satisfy both constraints together. Hence, above LP is feasible but unbounded.

iii. Not feasible.
Solution: Example of a not feasible LP is:

$$\begin{aligned} \text{Min } & x + y \\ \textbf{s.t. } & x \geq 8 \\ & x \leq 7 \\ & y \geq 0 \end{aligned}$$

Here, the constraint for x is so tight, that it is impossible to satisfy both constraints together. Hence, above LP is not feasible.

3. Prove that every problem in P is also in NP.
Proof: **NP** *is the set of decision problems that have polynomial − time* verifiers. Specificially, *problem Q is in* **NP** *if there is a polynomial − time algorithm* V(*I,X*) such that:

- *If I is a YES-instance, then there exists X such that V (I,X) = YES.*
- *If I is a NO-instance, then for all X, V (I,X) = NO.*

Furthermore, X should have length polynomial in size of I (since we are really only giving V time polynomial in the size of the instance, not the combined size of the instance and solution).
The second input X to the verifier V is often called a witness. E.g., for 3-coloring, the witness that an answer is YES is the coloring. For factoring, the witness that N has a factor between 2 and k is a factor. For the Traveling Salesman Problem: Given a weighted graph G and an integer k, does G have a tour that visits all the vertexes and has total length at most k? the witness is the tour. Because some **NP**-complete problems are dependent upon SAT to produce a deterministic polynomial time solution for them, and because SAT does not have a deterministic polynomial time solution, then P is a proper subset of NP. All these problems belong to **NP**. Of course, any problem in **P** is also in **NP**, since V could just ignore X and directly solve I.
So, **P** $\subseteq$ **NP**. *Q.E.D*

4. Suppose you are helping to organize a summer sports camp, and the following problem comes up: the camp is supposed to have at least one counselor whos skilled in each of the n sports covered by the camp. They have received job applications from m potential counselors. For each of the n sports, there is some subset of the m applicants qualified in that sport. The question is: for a given number $k < m$, is it possible to hire at most k of the counselors and have at least one counselor qualified in each of the n sports? We call this the *Efficient Recruiting* problem.

   - Show that *Efficient Recruiting* is NP-Complete.

   Solution: We know that, a problem Q is NP-complete if:

   (a) $Q \in NP$, and

   (b) Any other Q' in NP is polynomial-time reducible to Q; that is, $Q' \leq_p Q$.
   If Q just satisfies (b) then its called NP-hard.

   In order to prove that *Efficient Recruiting* is NP-Complete, we need to show that:

   (a) *Efficient Recruiting* is in NP, i.e, a polytime-checker exists for it.
   and

   (b) *Efficient Recruiting* is NP-hard, i.e., a known NP-complete problem can be reduced to this problem in polynomial time.

   (a) ***Efficient Recruiting* is in NP:**
   We can verify that the given solution of k counselors cover all n sports by going through the skills of each counselor and making the unique list of sports covered. This list can then be used to check whether all n sports are covered or not.
   This will take at-most $k * n$ running time.
   Thus, in polynomial time ($O(k * n)$), we can check if at least one counselor is qualified in each of the n sports.
   Hence, *Efficient Recruiting* is in NP.

   (b) ***Efficient Recruiting* is NP-hard:**
   We know that *Vertex Cover* is a known NP-complete problem.
   A vertex cover in a graph is a set of nodes such that every edge is incident to at least one of them.
   In *Vertex Cover* problem, we need to find a subset of vertexes of size at most a given number k that cover all the edges for the given a graph $G(V, E)$.
   Now, we reduce this problem to an *Efficient Recruiting* problem by assigning each vertex to a counselor and each edge to a sport.
   Therefore, if a counselor has skills for a specific sport, then there will be an edge incident on its vertex. and both the endpoints of the connecting edge represents the counselors who are experts in that sport.
   This construction of *Efficient Recruiting* from *Vertex Cover* takes polynomial

amount of time.

Now, to prove the correctness, we can consider a black box for the *Efficient Recruiting* and pass the result to *Vertex Cover* as it is.

If the black box for *Efficient Recruiting* returns a set of at most k counselors such that at least one counselor is qualified in each of the n sports, then it implies that every sport edge is incident on the subset of k counselors. This is nothing but the vertex cover of size k.

Also, if the vertex cover problem returns a vertex cover of size k for a graph, then, the subset of counselors corresponding to those vertexes will cover all the n sports.

Suppose that we can find , then we will find at most k vertexes that cover all the edges.

That is, we can solve vertex cover if we can solve Efficient recruiting problem in polynomial time and vice-versa.

Hence, Vertex Cover $\leq_p$ Efficient Recruiting as construction and black-box calls take polynomial time.

Thus, we have proved that *Efficient Recruiting* is in NP and it is also NP-hard.
Therefore, *Efficient Recruiting* is NP-complete.

5. The Satisfiability Problem (SAT) is the problem of determining if there exists a boolean assignment $(x_1, x_2, \cdots, x_n) \in 0, 1^n$ that satisfies a given Boolean formula. In Boolean logic, a formula is in conjunctive normal form (CNF) or clausal normal form if it is a conjunction of clauses, where a clause is a disjunction of literals. For example,

$$(x_1 \vee x_4 \vee \bar{x_5} \vee \bar{x_7}) \wedge (x_2 \vee \bar{x_4} \vee \bar{x_5}) \wedge \cdots \wedge (\bar{x_7} \vee x_3 \vee x_4 \vee x_5)$$

is a CNF-formula. In contrast, a disjunctive normal form (DNF) is a standardization (or normalization) of a logical formula which is a disjunction of conjunctive clauses. For example,

$$(x_1 \wedge x_4 \wedge \bar{x_5} \wedge \bar{x_8}) \vee (x_2 \wedge \bar{x_4} \wedge \bar{x_5}) \vee \cdots \vee (\bar{x_7} \wedge x_3 \wedge x_4 \wedge x_5)$$

is a DNF-formula.
Decide whether each of the following statements is correct and give a proof for each part.

(a) The SAT problem on CNF-formula instances is in NP.
Solution: The given statement is true.
A general CNF-formula is in NP but it may or may not be NP-complete. The SAT problem on CNF-formula instances is NP-complete if it is 3-CNF(at most 3 literals in each clause) but we can solve 2-CNF(at most 2 literals) in polynomial-time.
Since, every problem in P is also in NP, we can say that the SAT problem on CNF-formula instances is in NP.
Proof: *CNF-SAT* is in NP:
In order to show that a problem is in NP, we can prove that a poytime-checker exists for that problem.
For CNF-SAT, we can verify whether the given assignment is satisfiable or not by just substituting the literal values and computing boolean OR and boolean AND as per the CNF-formula.
This will take linear time for the entire computation and determine the satisfiability.
Hence, a polytime-checker exists for CNF-SAT.
Thus, *CNF-SAT* is in NP. Q.E.D.

(b) The SAT problem on DNF-formula instances is in P.
Solution: As mentioned, a disjunctive normal form (DNF) is a standardization (or normalization) of a logical formula which is a disjunction of conjunctive clauses. For example,

$$(x_1 \wedge x_4 \wedge \bar{x_5} \wedge \bar{x_8}) \vee (x_2 \wedge \bar{x_4} \wedge \bar{x_5}) \vee \cdots \vee (\bar{x_7} \wedge x_3 \wedge x_4 \wedge x_5)$$

is a DNF-formula.
Thus, it is boolean OR of boolean AND clauses. A boolean OR is satisfiable(true)

iff any one of its clause is satisfied(true).
Now, a single clause is boolean AND of literals. Any single AND clause is satisfiable for some values of its literals unless it consist of a literal and it's negation in the same clause (For e.g.: $(\bar{x} \wedge x)$).
Since, a DNF is OR of boolean AND clasues, the SAT-problem, would iterate over all the clauses till a satisfiable clause is found.
Thus the worst-case total running time would be equal to
(time required for processing a single clause) * (number of clauses in DNF).
Therefore, the total time will be polynomial. Hence, the SAT problem on DNF-formula instances is in P. Q.E.D.

6. **Dwarf Dormitories, Again!** Recall the problem from PS4's programming assignment. In this problem, you were given a grid of $R \times C$ elements, where C was a small constant, and were asked to find the maximum possible sum of elements, such that no two chosen elements are directly adjacent vertically or horizontally.
Suppose that C is no longer a small number. Let the grid be $N \times N$ elements. Give an algorithm that is polynomial in N, and returns a subset of elements which have maximum possible sum, constrained so that no two elements are directly adjacent vertically or horizontally.
Solution: Algorithm Idea: From the given inputs, we can construct a graph by taking all cells as the vertexes of the graph and adding a edge between two vertexes if they are adjacent horizontally or vertically. This will create a bipartite graph and we can then convert this graph to flow network. In order to find maximum possible sum of quality measure, we will compute maximum weighted independent set by computing minimum vertex cover using max-flow algorithm.
Detailed Steps: In the given problem, we have a $N \times N$ grid of cells.
Each cell has a quality measure q_{ij} associated to it.
Our goal is to find the subset of cells which have maximum possible sum of q_{ij} such that no two elements are directly adjacent horizontally or vertically.
This is nothing but a special case of maximum weighted independent set problem. We know that minimum weight vertex cover is complement of maximum weighted independent set problem and vice-versa.
Also, by Kőnig's theorem, the number of edges in a maximum matching equals the number of vertices in a minimum vertex cover for a bipartite graph.
We know that the minimum weight vertex cover can be solved easily by using max-flow/min-cut algorithm.
Hence, we can solve the given problem by reducing it to min-cut problem.

Let us construct a bipartite graph $G = (V, E)$ by taking all the cells as the vertexes of the graph.
We can add edges between two vertexes if they share a common edge, i.e., if they are adjacent horizontally or vertically.
This will split the graph into two parts, a left part and a right part.
Hence, the number of vertexes in the graph is $|V| = N^2$ as it is $N \times N$ grid and the number of edges in the graph is $|E| = 2N(N-1)$ as only adjacent vertexes are connected.
Now, in order to apply max-flow algorithm, we will convert this graph to a flow network by adding a super source vertex S which will be connected to all left side vertexes of the bipartite graph.
And a super sink node T which will be connected to all right side vertexes of the bipartite graph.
Now, we will add the capacities to the edges as follows:
From source S to all left vertexes, the edge capacity would be equal to the quality

measure of that vertex, i.e., q_{ij}.
From all right vertexes to the sink T, the edge capacity would be equal to the quality measure of that vertex, i.e., q_{ij}.
Also, the capacities for the edges connecting left and right vertexes of the original graph is ∞.
Now, this is converted to a flow network where the edge capacities have integer values.
Hence, to find the minimum S-T cut for the above flow network, we can use Edmonds-Karp network flow algorithm.
The value of this min-cut is equivalent to the minimum vertex cover.
In order to retrieve the minimum vertex cover, we can add use all the vertexes that are adjacent to cut-edges or use the set of left vertexes which are not reachable from source S and the right vertexes which are reachable from source S.

Correctness: Since, the capacity of original edges is ∞ , the min-cut will not contain those edges. Hence, the final set of vertices will not have any vertex that is adjacent horizontally or vertically. Aslo, if we cut a left-edge, then corresponding vertex is added to minimum vertex cover set. Otherwise, all the right-edges which are adjacent to this vertex will be cut. This ensures correctness.

Complexity: For the graph $G = (V, E)$, the number of vertexes is equal to all reservoirs, source and sink, i.e., $|V| = (N^2 + 2)$. For a sufficiently large value of N, we can approximate $(N^2 + 2)$ to N^2.
Also, the number of edges is equal to $|E| = 2N(N-1) \approx O(N^2)$.
Hence, in order to construct the graph $G = (V, E)$, the worst-case complexity is $O(N^4)$.
Since, we have used Edmonds-Karp algorithm to calculate max-flow, the worst case complexity is $O(|V||E|^2)$.
By substituting the above values of $|V|$ and $|E|$, we get the complexity as $O(N^2 * N^4)$
Hence, the worst case complexity is $O(N^6)$.

2 Part B (programming)

1. **Linear Programming** In a programming language of your choice, write a single program to take descriptions of linear programs as input, and outputs solutions to the linear program. You may use an external library for solving linear programs. Submit your code as a single file.
   For each of the following problems, give a formal representation of the linear program, convert it into input to your solver, and give the resulting output. Give both your converted inputs and resulting outputs as part of your submission:

   Program: Given below is the source code of our LP solver that takes input from stdin, uses cplex library to solve the problem and writes the output to stdout.

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;

import ilog.concert.*;
import ilog.concert.IloCopyManager.Check;
import ilog.cplex.*;

public class LPSolver {

   public static void main(String[] args) throws IOException
   {
      BufferedReader br = new BufferedReader(new
          InputStreamReader(System.in));

      String[] s = br.readLine().split(" ");

      int n = Integer.parseInt(s[0]);
      int c = Integer.parseInt(s[1]);

      s = br.readLine().split(" ");

      double[] objvals = new double[n];
      double[] lb = new double[n];
      double[] ub = new double[n];

      // set bounds

      for (int i = 0; i < n; i++)
      {
         objvals[i] = Double.valueOf(s[i]);
```

```
   lb[i] = 0.0;
   ub[i] = Double.MAX_VALUE;
}

// Bounds Set

// Scan for constraints

int constraints [][] = new int[c][n + 1];

for (int i = 0; i < c; i++)
{
   s = br.readLine().split(" ");

   for (int j = 0; j < n + 1; j++)
   {
      constraints [i][j] = Integer.parseInt(s[j]);
   }
}

// Constraints done. Solve

try
{
   IloCplex cplex = new IloCplex();
   IloNumVar[] x = cplex.numVarArray(n, lb, ub);

   cplex.addMaximize(cplex.scalProd(x, objvals));

   for (int i = 0; i < c; i++)
   {
      IloNumExpr curr_expr[] = new IloNumExpr[n];

      for (int j = 0; j < n; j++)
      {
         curr_expr[j] = cplex.prod(constraints[i][j], x[j]);
      }

      cplex.addLe(cplex.sum(curr_expr), constraints[i][n]);
   }

   if (cplex.solve())
   {
      // Printing as double casted to long for understanding of
          absolute values
```

```
            System.out.println("Maximized Value = " + (long)
                cplex.getObjValue());
            double[] val = cplex.getValues(x);
            int ncols = cplex.getNcols();
            for (int j = 0; j < ncols; ++j)
            {
               // Printing as double casted to long for understanding of
                  absolute values
               System.out.println("Objective Value of the element " + j + " :
                  " + (long) val[j]);
            }
         }
         cplex.end();
      }
      catch (IloException e)
      {
         System.err.println("Concert exception '" + e + "' caught");
      }
   }
}
```

(a) A factory can produce three types of widgets - A, B and C. There are 100 assembly lines in the factory and each assembly line has to be dedicated to one widget-type or it can be left unused. Each widget A assembly line requires $18 of supplies and 2 hours of labor. Each widget B assembly line requires $36 of supplies and 6 hours of labor. And, each widget C assembly line requires $24 of supplies and 4 hours of labor. Labor costs are $8/hour. There is a total of $2100 for supplies and $2400 for labor. You know that each widget A assembly line generates $55 worth of revenue, each widget B assembly line generates $100 worth of revenue and each widget C assembly line generates $125 worth of revenue. Find the optimal configuration of assembly lines (i.e how many assembly lines for each type of widget) so as to maximize the factorys total revenue.
Solution: The above problem can be written in formal representation of linear programming as:

$$\begin{aligned} \text{Max } & 55A + 100B + 125C \\ \text{s.t. } & 18A + 36B + 24C \leq 2100 \\ & 2 * 8 * A + 6 * 8 * B + 4 * 8 * C \leq 2400 \\ & A + B + C \leq 100 \\ & A, B, C \geq 0 \end{aligned}$$

The reduced form is:

$$\begin{aligned} \text{Max } & 55A + 100B + 125C \\ \text{s.t. } & 3A + 6B + 4C \leq 350 \\ & A + 3B + 2C \leq 150 \\ & A + B + C \leq 100 \\ & A, B, C \geq 0 \end{aligned}$$

Input: Following is this problem's input file for the above generalized LP solver problem:

```
3 3 // number of elements and constraints
55 75 125 // Maximize objective
18 36 24 2100 // Constraints for supplies
16 48 32 2400 // Constraints for labor
1 1 1 100 // Constraints for assembly line units
```

Output: The output given by above LP solver program:

```
Maximized Value = 9375
Objective Value of the element 0 : 0
Objective Value of the element 1 : 0
Objective Value of the element 2 : 75
```

(b) In the country of Koodstopia, there are four major cities, labeled A, B, C, and D. Between every pair of cities there is a road. You would like to place tolls on the roads in order to maximize income for the government. However, if these tolls are too high, the citizens will revolt. Citizens do not communicate very well with citizens in different cities, so it turns out that you can avoid a revolt by making sure that the total toll amounts for roads in / out of every city are under a certain threshold.
The populations of cities A, B, C, and D are $1000, 2000, 2500$, and 3000. If you place a toll t on a road between two cities, the total profit will be t times the product of the populations of those cities. If a city with population p has more than p total tolls on all of the roads in / out of the city, the city population will revolt. Find the optimal assignment of tolls to roads so as to maximize the total income while avoiding a revolt.

Solution: The above problem can be written in formal representation of linear programming as:

$$\begin{aligned} \text{Max } (1000 * 2000)u + (1000 * 2500)v + \\ (1000 * 3000)w + (2000 * 2500)x + \\ (2000 * 3000)y + (2500 * 3000)z \\ \text{s.t. } u + v + w \leq 1000 \\ u + x + y \leq 2000 \\ v + x + z \leq 2500 \\ w + y + z \leq 3000 \\ u, v, w, x, y, z \geq 0 \end{aligned}$$

where :
we maximize objective that is product of tolls, AB, AC, AD, BC, BD, DC
u: road between city A and B
v: road between city A and C
w: road between city A and D
x: road between city B and C
y: road between city B and D
z: road between city C and D

Input: Following is this problem's input file for the above generalized LP solver problem:

```
6 4 // number of elements and constraints
2000000 2500000 3000000 5000000 6000000 7500000 // Maximize objective function
1 1 1 0 0 0 1000 // Constraints as per city A population limit
1 0 0 1 1 0 2000 // Constraints as per city B population limit
0 1 0 1 0 1 2500 // Constraints as per city C population limit
0 0 1 0 1 1 3000 // Constraints as per city D population limit
```

Output: The output given by above LP solver program:

```
Maximized Value = 24625000000
Objective Value of the element 0 : 1000
Objective Value of the element 1 : 0
Objective Value of the element 2 : 0
Objective Value of the element 3 : 250
Objective Value of the element 4 : 750
Objective Value of the element 5 : 2250
```

***Note:** We have removed the auto-generated console debug statements of cplex solver and have just adding the stdout statements generated by our code.*