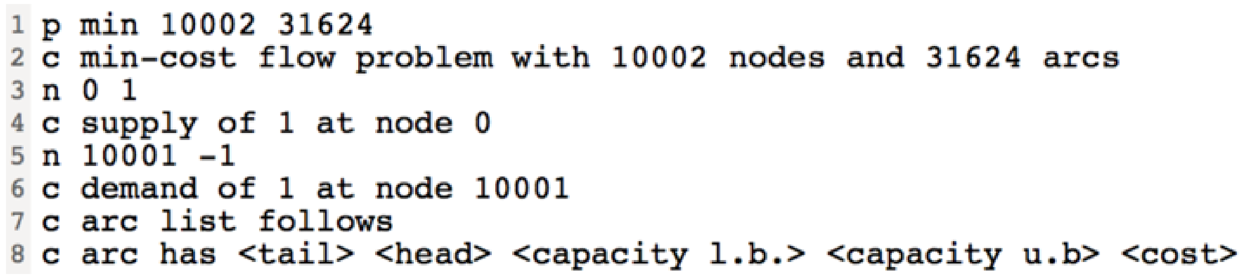
Anna Sun, Chen Bai, Shiyu Wang

Algorithm PS 4

1. According to the problem, suppose a graph $G=(V,E)$. There are 10000 nodes in G represented by $1,2,\ldots,10000$ and there is a directed arc from i to j if i/j is the form of $2^{\alpha}3^{\beta}(\alpha,\beta \in N)$. The capacity of each arc is 1 and the cost is -1 so that we can solve this problem by finding the min-cost flow of G.

1) Add a source 0 and a sink 10001, both the supply of the source and the the demand of the sink are 1.
2) Add an arc from 0 to those nodes which is a starting node of an arc in G. The capacity of these arcs is 1 and the cost is 0.
   i.e *If* $i = 24$ & $j = 1$, $i/j = 2^3 3^1$. *So add an arc from* 0 *to* 24.
3) Add an arc from those nodes which is a ending node of an arc in G to 10001. The capacity of these arcs is 1 and the cost is 0.
   i.e *If* $i = 144$ & $j = 3$, $i/j = 2^4 3^1$. *So add an arc from* 3 *to* 10001.

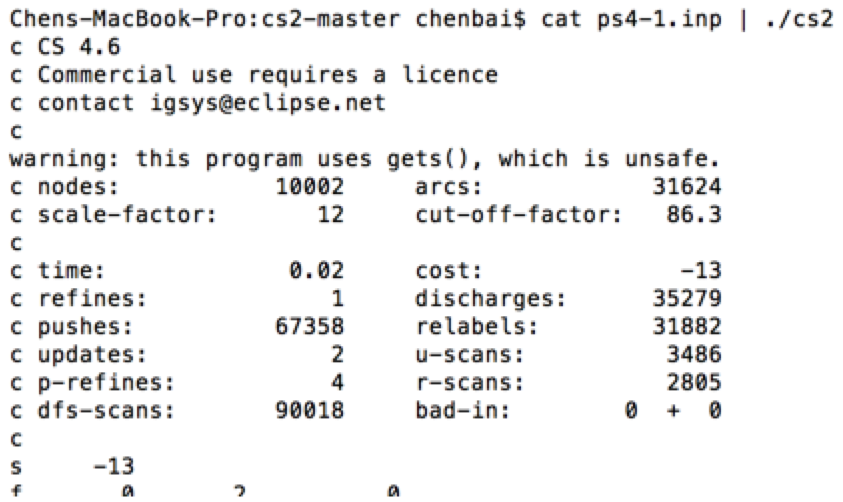
Following is the description of this problem in the input file.



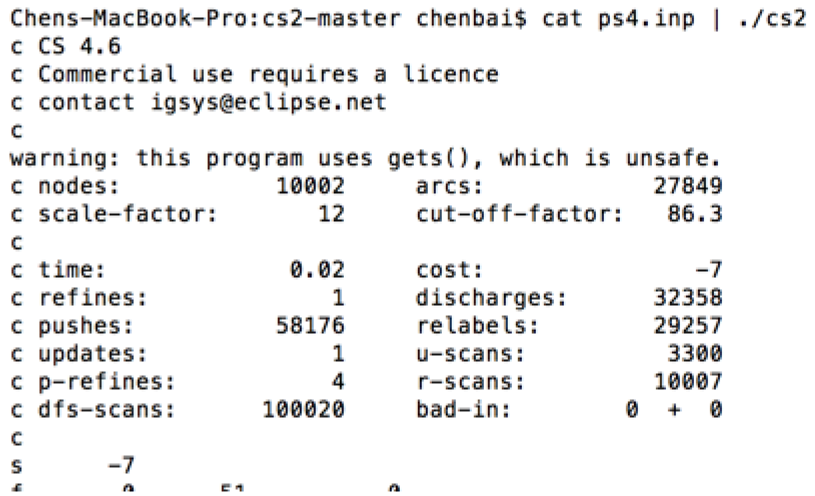
```
p min 10002 31624
c min-cost flow problem with 10002 nodes and 31624 arcs
n 0 1
c supply of 1 at node 0
n 10001 -1
c demand of 1 at node 10001
c arc list follows
c arc has <tail> <head> <capacity l.b.> <capacity u.b> <cost>
```

Thus, we need to run the program at most 27 times to get the max number of nodes covered by node-disjoint paths. For each time, we must remove the nodes and arcs which have been covered.

Following is the result for the first and the last running.



```
Chens-MacBook-Pro:cs2-master chenbai$ cat ps4-1.inp | ./cs2
c CS 4.6
c Commercial use requires a licence
c contact igsys@eclipse.net
c
warning: this program uses gets(), which is unsafe.
c nodes:            10002     arcs:            31624
c scale-factor:        12     cut-off-factor:   86.3
c
c time:              0.02     cost:              -13
c refines:              1     discharges:      35279
c pushes:           67358     relabels:        31882
c updates:              2     u-scans:          3486
c p-refines:            4     r-scans:          2805
c dfs-scans:        90018     bad-in:        0  +  0
c
s     -13
f       0       2          0
```



```
Chens-MacBook-Pro:cs2-master chenbai$ cat ps4.inp | ./cs2
c CS 4.6
c Commercial use requires a licence
c contact igsys@eclipse.net
c
warning: this program uses gets(), which is unsafe.
c nodes:            10002     arcs:            27849
c scale-factor:        12     cut-off-factor:   86.3
c
c time:              0.02     cost:               -7
c refines:              1     discharges:      32358
c pushes:           58176     relabels:        29257
c updates:              1     u-scans:          3300
c p-refines:            4     r-scans:         10007
c dfs-scans:       100020     bad-in:        0  +  0
c
s      -7
f       0      51          0
```

The covered nodes for each path:

1)	1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192	min-cost: -13
2)	3, 9, 18, 36, 72, 144, 288, 576, 1152, 2304, 4608, 9216	min-cost: -11
3)	5, 10, 20, 40, 80, 160, 480, 960, 1920, 3840, 7680	min-cost: -10
4)	7, 14, 28, 56, 112, 224, 448, 896, 1792, 3584, 7168	min-cost: -10
5)	6, 12, 24, 48, 96, 192, 384, 768, 1536, 3072, 6144	min-cost: -10
6)	11, 22, 44, 88, 176, 528, 1056, 2112, 4224, 8448	min-cost: -9
7)	19, 38, 76, 152, 304, 608, 1216, 2432, 4864, 9728	min-cost: -9
8)	13, 26, 52, 104, 208, 416, 832, 1664, 4992, 9984	min-cost: -9
9)	17, 34, 68, 136, 272, 544, 1088, 2176, 4352, 8704	min-cost: -9
10)	21, 42, 84, 168, 336, 672, 1344, 2688, 8064	min-cost: -8
11)	15, 45, 90, 180, 540, 1080, 2160, 4320, 8640	min-cost: -8
12)	29, 58, 116, 232, 464, 928, 1856, 3712, 7424	min-cost: -8

13) 23, 46, 92, 184, 368, 736, 1472, 2944, 8832	min-cost: -8
14) 31, 62, 124, 248, 496, 992, 1984, 3968, 7936	min-cost: -8
15) 35, 70, 140, 280, 560, 1120, 2240, 4480, 8960	min-cost: -8
16) 27, 54, 108, 216, 432, 864, 1728, 3456, 6912	min-cost: -8
17) 37, 74, 148, 296, 592, 1184, 2368, 4736, 9472	min-cost: -8
18) 25, 75, 150, 300, 600, 1200, 2400, 4800, 9600	min-cost: -8
19) 71, 142, 284, 568, 1136, 2272, 4544, 9088	min-cost: -7
20) 77, 154, 308, 616, 1232, 2464, 4928, 9856	min-cost: -7
21) 67, 134, 268, 536, 1072, 2144, 4288, 8576	min-cost: -7
22) 65, 130, 260, 520, 1040, 2080, 4160, 8320	min-cost: -7
23) 73, 146, 292, 584, 1168, 2336, 4672, 9344	min-cost: -7
24) 53, 106, 212, 424, 848, 1696, 3392, 6784	min-cost: -7
25) 41, 82, 164, 492, 984, 1968, 3936, 7872	min-cost: -7
26) 50, 100, 200, 400, 800, 1600, 3200, 6400	min-cost: -7
27) 30, 60, 120, 240, 720, 1440, 2880, 5760	min-cost: -7

As a result, the maximum number of nodes (numbers) covered by at most 27 node-disjoint paths is 252.

2.(a)

i)Canonical Form:

Objective function can be rewritten as:

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

Replace the first constraint by two inequalities: $2y - 4z \le 7$ & $-2y + 4z \le -7$. The second constraint can be written as: $2y - 4x \le -5$. There is no need to rewrite the third constraint. Decision variable z can be expressed by introducing two extra nonnegative variables as:

$$z = z' - z''$$

Thus, z can be negative if $z' < z''$ and positive if $z' > z''$ depending on the values of z' and z''. z can be zero also if $z' = z''$. Thus, the standard form of this problem is as follows:

$$\begin{aligned} \text{Max} \;\; & 4(z' - z'') - 5x - 3y \\ \text{s.t.} \;\; & 2y - 4(z' - z'') \le 7 \\ & -2y + 4(z' - z'') \le -7 \\ & 2y - 4x \le -5 \\ & 3x + 5(z' - z'') \le -3 \\ & x, y, z', z'' \ge 0 \end{aligned}$$

ii)Standard Form:

Objective function can be rewritten as:

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

The first constraint is an equation, there is nothing to change. The second constraint can be rewritten as $4x - 2y - u = 5$. Note that, a new nonnegative variable u is added to the left-hand-side (LHS) to make both sides equal. Similarly, the third constraint can be rewritten as: $-3x - 5z - v = 3$. The variables u and v are known as surplus variables.
Decision variable z can be expressed by introducing two extra nonnegative variables as:

$$z = z' - z''$$

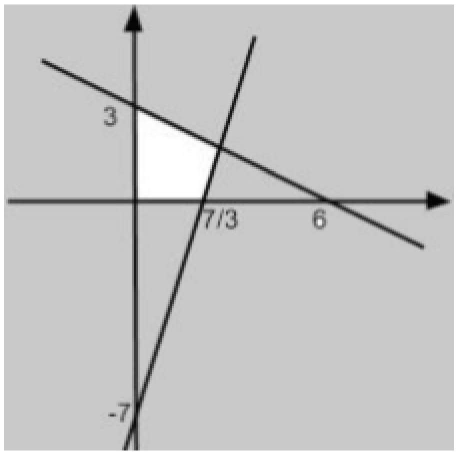
Thus, z can be negative if $z' < z''$ and positive if $z' > z''$ depending on the values of z' and z''. z can be zero also if $z' = z''$. Thus, the canonical form of this problem is as follows:

$$\begin{gathered}\text{Max } \; 4(z' - z'') - 5x - 3y \\ \text{s.t. } \; 2y - 4(z' - z'') = 7 \\ 4x - 2y - u = 5 \\ -3x - 5z - v = 3 \\ x, y, z', z'', u, v \geq 0\end{gathered}$$

(b)
i)Feasible and bounded

$$\begin{gathered}\text{Max } \; 5x - 3y \\ 3x - y \leq 7 \\ x + 2y \leq 6 \\ x \geq 0, \; y \geq 0\end{gathered}$$

The feasible region for this set of constraints was shown above.

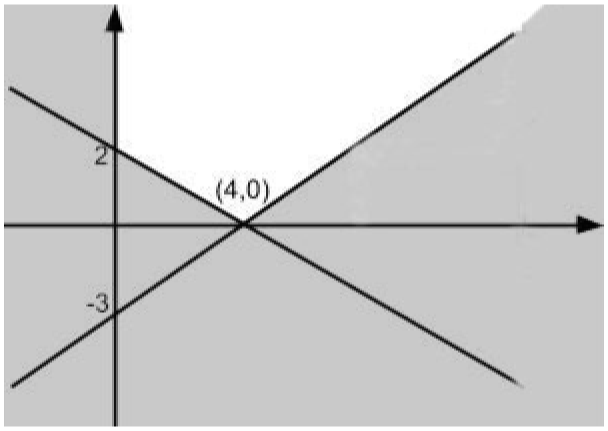


ii)Feasible and unbounded

$$\begin{gathered}\text{Min } \; 3x + 4y \\ \text{s.t. } \; 3x - 4y \leq 12 \\ x + 2y \geq 4\end{gathered}$$

$$x \geq 0,\ y \geq 0$$

The feasible region for this set of constraints was shown above.



3.Given: $m \times n$ matrix M

1) Row player selects a strategy $i \in \{1, \ldots, m\}$.
2) Col player selects a strategy $j \in \{1, \ldots, n\}$.
3) Row pays Col a_{ij} dollars.

Suppose Col player were to adopt strategy C. Then, Row player's best defense is to use R that minimizes $R^T Ax$:

$$\min_R R^T MC$$

And so Col player should choose that x which maximizes these possibilities:

$$\max_C \min_R R^T MC$$

Inner optimization is:

$$\min_R R^T MC = \min_i e_i^T MC$$

(e_i denotes the vector that's all zeros except for a one in the i-th position--that is, deterministic strategy i).

Now, we have:

$$max(\min_i e_i^T MC)$$

$$\sum_j C_j = 1$$

$$C_j \geq 0, j = 1, 2, \ldots,\ \ n$$

Introduce a scalar variable v representing the value of the inner minimization:

$$max\ v$$

$$v \leq e_i^T MC,\ i = 1, 2, \ldots, m$$

$$\sum_j C_j = 1$$

$$C_j \geq 0, j = 1, 2, \ldots, \quad n$$

Writing in pure matrix-vector notation:

$$\begin{gathered} max\ v \\ ve - MC \leq 0 \\ e^T C = 1 \\ C \geq 0 \end{gathered}$$

(e denotes the vector of all ones).

Similarly, Row player seeks y^* attaining:

$$\min_R \max_C R^T MC$$

which is equivalent to:

$$\begin{gathered} min\ u \\ ue - R^T M \geq 0 \\ eR^T = 1 \\ R \geq 0 \end{gathered}$$

(a)The linear program to compute the minimum payoff that the Row player can achieve without knowing the (mixed) strategy of the Col player is following:

$$\begin{gathered} min\ u \\ \text{s.t. } ue - R^T M \geq 0 \\ eR^T = 1 \\ R \geq 0 \end{gathered} \tag{3.1}$$

(b)The linear program to compute the maximum payoff that the Col player can achieve without knowing the (mixed) strategy of the Row player is following:

$$\begin{gathered} max\ v \\ \text{s.t. } ve - MC \leq 0 \\ e^T C = 1 \\ C \geq 0 \end{gathered} \tag{3.2}$$

(c)Proof:

We can rewrite LP(3.1) in Block Matrix Form:

$$\min \begin{bmatrix} 0 \\ 1 \end{bmatrix}^T \begin{bmatrix} R \\ u \end{bmatrix}$$

$$\begin{bmatrix} -M^T & e \\ e^T & 0 \end{bmatrix}^T \begin{bmatrix} R \\ u \end{bmatrix} \begin{matrix} \geq \\ = \end{matrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$R \geq 0$$

$$u \text{ free}$$

We can rewrite LP(3.2) in Block Matrix Form:

$$\max \begin{bmatrix} 0 \\ 1 \end{bmatrix}^T \begin{bmatrix} C \\ v \end{bmatrix}$$

$$\begin{bmatrix} -M & e \\ e^T & 0 \end{bmatrix} \begin{bmatrix} C \\ v \end{bmatrix} \begin{matrix} \leq \\ = \end{matrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix}$$

$$C \geq 0$$

$$v \text{ free}$$

Obviously, Row's problem is dual to Col's.

From Strong Duality Theorem, there is

$$u^* = v^*$$

Also,

$$v^* = \min_i e_i^T M C^* = \min_R R^T M C^*$$

$$u^* = \max_j R^{*T} M e_j = \max_C R^{*T} M C$$

Let C^* denote Col player's solution to his max-min problem. And let R^* denote Row player's solution to his min-max problem.

Then

$$\max_C R^{*T} M C = \min_R R^T M C^*$$

The method to establish Nash's theorem was given above.

4. Assume there are x_1 assembly lines to produce A, x_2 assembly lines to produce B, x_3 assembly lines to produce C.

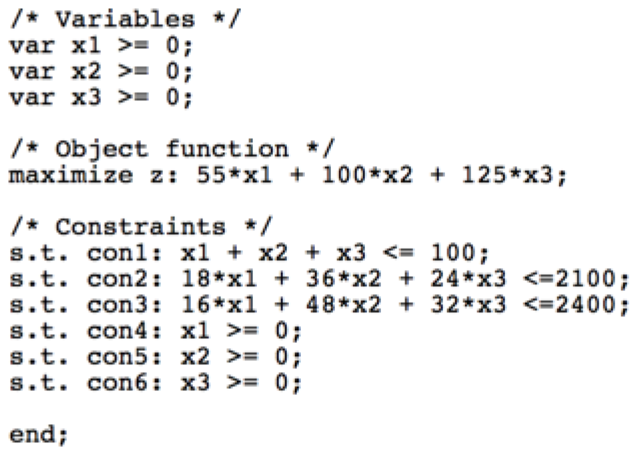
Objective function:

$$\text{Max } Z = 55x_1 + 100x_2 + 125x_3$$

Constraints:

$$\begin{aligned}\text{s.t. } x_1 + x_2 + x_3 &\le 100\\ 18x_1 + 36x_2 + 24x_3 &\le 2100\\ 16x_1 + 48x_2 + 32x_3 &\le 2400\\ x_1, x_2, x_3 &\ge 0\end{aligned}$$

Input file: glpsolEx.mod



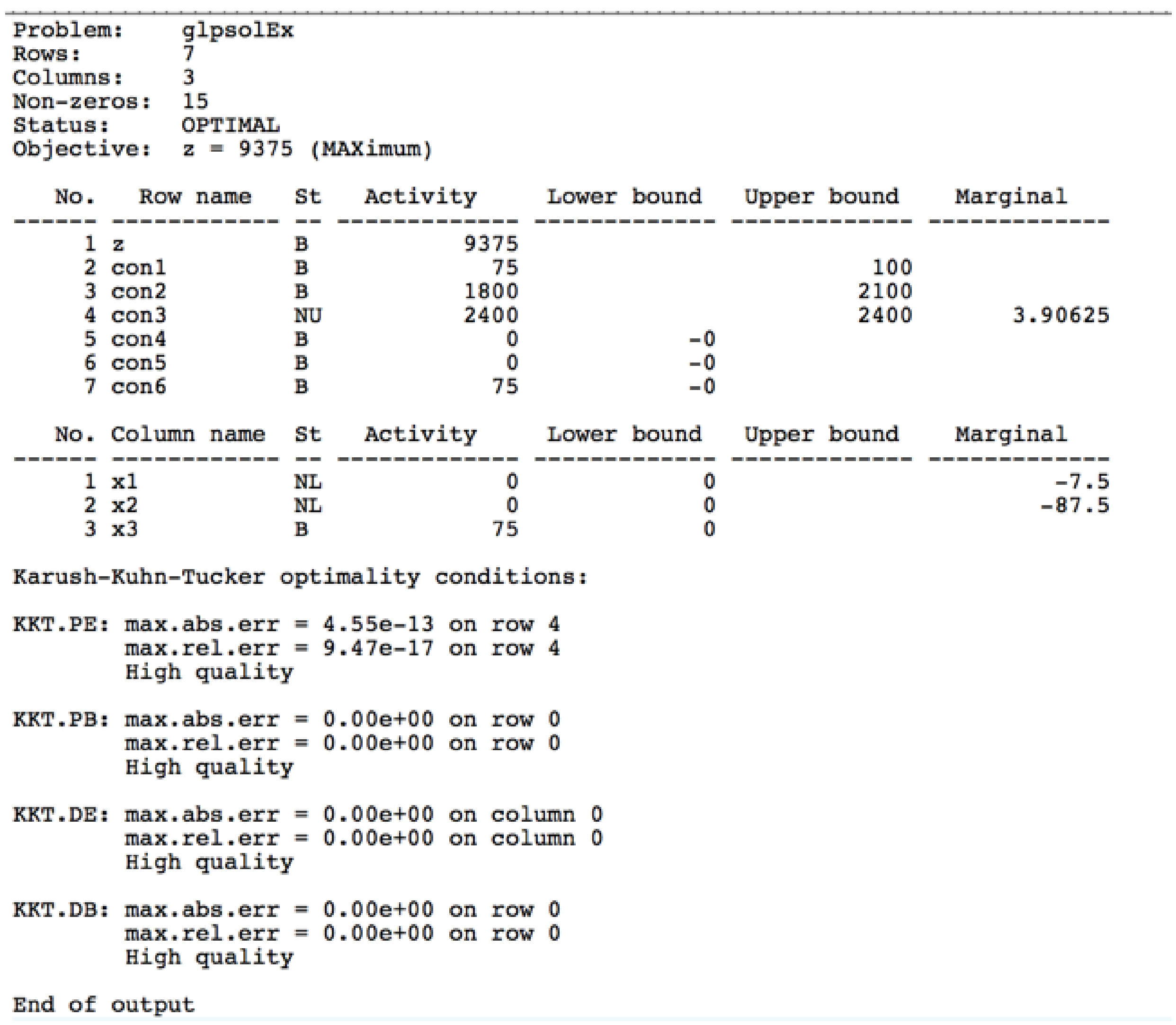
```
/* Variables */
var x1 >= 0;
var x2 >= 0;
var x3 >= 0;

/* Object function */
maximize z: 55*x1 + 100*x2 + 125*x3;

/* Constraints */
s.t. con1: x1 + x2 + x3 <= 100;
s.t. con2: 18*x1 + 36*x2 + 24*x3 <=2100;
s.t. con3: 16*x1 + 48*x2 + 32*x3 <=2400;
s.t. con4: x1 >= 0;
s.t. con5: x2 >= 0;
s.t. con6: x3 >= 0;

end;
```

Solution file: glpsolEx.sol



```
Problem:    glpsolEx
Rows:       7
Columns:    3
Non-zeros:  15
Status:     OPTIMAL
Objective:  z = 9375 (MAXimum)

   No.   Row name   St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
     1 z            B           9375
     2 con1         B             75                         100
     3 con2         B           1800                        2100
     4 con3         NU          2400                        2400       3.90625
     5 con4         B              0            -0
     6 con5         B              0            -0
     7 con6         B             75            -0

   No. Column name  St   Activity     Lower bound   Upper bound    Marginal
------ ------------ -- ------------- ------------- ------------- -------------
     1 x1           NL             0             0                        -7.5
     2 x2           NL             0             0                       -87.5
     3 x3           B             75             0

Karush-Kuhn-Tucker optimality conditions:

KKT.PE: max.abs.err = 4.55e-13 on row 4
        max.rel.err = 9.47e-17 on row 4
        High quality

KKT.PB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

KKT.DE: max.abs.err = 0.00e+00 on column 0
        max.rel.err = 0.00e+00 on column 0
        High quality

KKT.DB: max.abs.err = 0.00e+00 on row 0
        max.rel.err = 0.00e+00 on row 0
        High quality

End of output
```

5.
Proof:
Suppose there is a graph $G = (V, E)$, each node in V represents a countryman, and each edge in E connects two friends.
According to the problem, we need to find the minimum number of nodes to dominate all nodes of the graph G. The set of nodes is the solution to this problem.
Obviously, the problem is to find the min dominating set of the graph G. We know that the vertex cover problem is NP-complete. Thus, we have to prove that dominating set is NP-complete by reduction from the vertex cover problem.

To prove that dominating set is in NP, we need to prove that given a set V_S, we can verify that it's a dominating set in polynomial time. This is easy to do:

a. For each vertex $\in V_S$, remove this vertex and all incidents vertices from G.

b. Check if all vertices were removed from G.

Reduction:
Create a new graph G' such that there exists a vertex for each vertex in G. Furthermore, add a new vertex for each edge in G, such that there now exists a triangle of edges. That's to say, for an edge, $e=(u,v)$, there now exists a new vertex, w, and the edges $(u,v),(u,w),(v,w)$.

Now, We have to prove that the graph G has a vertex cover of size k iff the new graph G' has a dominating set of size k. Assume we have a set, S, that is a vertex cover of G, then we know that the same set is a dominating set on G'. By definition of vertex cover, each edge in G is incident to at least one vertex in S. Therefore, each triangle of vertices in G' has at least one member of which belongs to S. As a result, each vertex in G' is either in S or adjacent to S. Thus, the dominating set in G' is a vertex cover in G.

Hence, this problem is NP-complete.

6.
Idea: Define a function f that converts instances C of Circuit-SAT to instances of 3-SAT such that the formula f(C) produced is satisfiable iff the circuit C had an input x such that C(x) = 1. Moreoverm f(C) should be computable in polynomial time, which among other things means we cannot blow up the size of C by More than a polynomial factor.

Reduction:
1. Assume the input is given as a list of gates,for each gate g_i:
   a. We are told what its inputs are connected to.
   b. In addition, which gate g_m is the output of the circuit
2. Compile this into an instance of 3-SAT. Make one variable for each input x_i of the circuit, and one for each gate g_i.
3. Write each NAND as a conjunction of 4 clauses. In particular, we just replace each statement of the form $y_3 = \text{NAND}(y_1, y_2)$ with:

$$\begin{array}{lll} & (y_1 \text{ or } y_2 \text{ or } y_3) & \leftarrow \text{if } y_1 = 0 \text{ and } y_2 = 0 \text{ then we must have } y_3 = 1 \\ \text{AND} & (y_1 \text{ or } \neg y_2 \text{ or } y_3) & \leftarrow \text{if } y_1 = 0 \text{ and } y_2 = 1 \text{ then we must have } y_3 = 1 \\ \text{AND} & (\neg y_1 \text{ or } y_2 \text{ or } y_3) & \leftarrow \text{if } y_1 = 0 \text{ and } y_2 = 1 \text{ then we must have } y_3 = 1 \\ \text{AND} & (\neg y_1 \text{ or } \neg y_2 \text{ or } \neg y_3) & \leftarrow \text{if } y_1 = 1 \text{ and } y_2 = 1 \text{ then we must have } y_3 = 0 \end{array}$$

4. Add the caluse (g_m), requireing the circuit to output 1. So, the 3-CNF formula produced is satisfiable if and only if the circuit has a setting of inputs that causes it to output 1. The size of the formula is linear in the size of the circuit. Moreover, the

construction can be done in polynomial (actually, linear) time. So, if we had a polynomial-time algorithm to solve 3-SAT, then we could solve circuit-SAT in polynomial time too. Done