**Homework #5**
**Solution Manual**

# Problem 1 Puzzle (programming assignment)

```
#include <stdio.h>
int lim,tbl[4][4];
int heuristic() {
        int ans=0;
        for ( int i=0; i<4; i++ ) for ( int j=0; j<4; j++ ) {
                int x=tbl[i][j]/4;
                int y=tbl[i][j]%4;
                ans+=(x-i+4)%4+(y-j+4)%4;
        }
        return (ans+3)/4;
}
void move_row( int r, int d ) {
        int tmp[4];
        for ( int i=0; i<4; i++ ) tmp[(i+d)%4]=tbl[r][i];
        for ( int i=0; i<4; i++ ) tbl[r][i]=tmp[i];
}
void move_col( int c, int d ) {
        int tmp[4];
        for ( int i=0; i<4; i++ ) tmp[(i+d)%4]=tbl[i][c];
        for ( int i=0; i<4; i++ ) tbl[i][c]=tmp[i];
}
int path[100];
int dfs( int lv, int pre ) {
        int h=heuristic();
        if ( lv+h>lim ) return 0;
        if ( h==0 ) return 1;
        for ( int i=pre<4?pre:0; i<4; i++ ) {
                path[lv]=i;
                move_col(i,1);
                if ( dfs(lv+1,i) ) return 1;
                move_col(i,3);
        }
        for ( int i=pre>=4?pre-4:0; i<4; i++ ) {
                path[lv]=i+4;
                move_row(i,1);
                if ( dfs(lv+1,i+4) ) return 1;
                move_row(i,3);
        }
        return 0;
}
int main()
{
        for ( int i=0; i<4; i++ ) for ( int j=0; j<4; j++ ) scanf("%X",tbl[i]+j);
        while ( !dfs(0,0)  ) lim++;
        printf("%d\n",lim);
        for ( int i=0; i<lim; i++ ) printf("%c%d\n",path[i]<4?'C':'R',path[i]%4);
        return 0;
}
```

For each state, we can estimate its lower bound of steps to achieve goal state, called $h$, by calculating the distance to goal state for each cell separately. Since we will change 4 cells in one step, the sum of distance dividing by 4 is a good estimation. If the number of step we have taken, called $g$, such that $g + h$ is larger than the answer limit we set, it is impossible to achieve goal state within the limit. By relaxing the limit one by one with proper pruning, and search in lexicographic order, we will obtain the optimal solution with smallest lexicographic order. A good pruning here is do not search the redundant states like moving $C2$ then $C1$, which is equivalent to $C1$ then $C2$, but has smaller lexicographic order. Note that we can read the initial state by scanf with "%X".

# Problem 2 New Policemans Confusion

1) Treat the straight road as a number line, and the police station located in 0, the speed of policeman and thief be 2 and 1 unit(s) per second respectively. Let the policeman run in maximum speed to the position $(+1, -4, +16, \ldots)$ until catching the thief, which will always catch the thief in finite time as the analysis in the next problem.

2) Without loss of generality, we can assume that the thief move in one direction with full speed. Let the position sequence $(+1, -4, +16, \ldots)$ be $p$, where $p_i = (-4)^i$.

   If the thief move in positive direction, consider the smallest $T' \geq T$ such that $T' = 4^{2k}$ for some non-negative integer $k$, the thief need to take $T'$ seconds to position $T'$, but the policeman only need to take

$$\frac{4^{2k}}{2} + \sum_{i=0}^{2k-1} 4^i = \frac{4^{2k}}{2} + \frac{4^{2k}-1}{4-1} \leq \frac{4^{2k}}{2} + \frac{4^{2k}}{3} \leq 4^{2k} = T'$$

   seconds, which means the thief is caught by the policeman.

   If the thief move in negative direction, consider the smallest $T' \geq T$ such that $T = 4^{2k+1}$ for some non-negative integer $k$, the thief need to take $T'$ seconds to position $-T'$, but the policeman only need to take

$$\frac{4^{2k+1}}{2} + \sum_{i=0}^{2k} 4^i = \frac{4^{2k+1}}{2} + \frac{4^{2k+1}-1}{4-1} \leq \frac{4^{2k+1}}{2} + \frac{4^{2k+1}}{3} \leq 4^{2k+1} = T'$$

   seconds, which also means the thief is caught by the policeman.

   As above, we know that the policeman only need not more than $T' \leq 256T = O(T)$ seconds to catch the thief!

3) No we can not, since we only know that the thief did not stole the car within $T$ seconds ago for some finite $T$, we do not even know whether the thief stole the car $T + 1$ seconds ago.

# Problem 3 NP's Closure

1) Since $L_a \in \mathrm{P}$, Let $A$ decides $L_a$ in polynomial time, simply make a new algorithm $A'$ which run and reverse the result of $A$, which will be a polynomial time algorithm.

2) Since $L_a, L_b \in \mathrm{P}$, Let $A, B$ decide $L_a, L_b$ in polynomial time respectively, simply make a new algorithm $C$ which accept $x = x_1 x_2 \cdots x_n$ iff there is a split position $p$, such that $A$ accept $x_1 \cdots x_p$ and $B$ accept $x_{p+1} \cdots x_n$. The algorithm $C$ only need to run $O(n)$ times $A$ and $B$, so $C$ decides $L_a L_b$ in polynomial time.

3) Since $L_a, L_b \in \mathrm{NP}$, Let $A, B$ verify $L_a, L_b$ in polynomial time respectively, simply make a new algorithm $D$ which accept $x, y$ iff $A$ or $B$ accept $x, y$, which will be a polynomial time algorithm.

4) Since $L_a \in \mathrm{NP}$, Let $A$ verify $L_a$ in polynomial time, simply make a new algorithm $E$ which accept $x, y$ iff we can factor $x, y$ to $x = x_1 x_2 \cdots x_n$ and $y_1, y_2, \cdots, y_n$ by some predefined rule, such that $A$ accept all $(x_i, y_i)$. It is easy to construct a rule which make the new certificate $y$ still satisfied the constraint $|y| = O(|x|^c)$ for some constant $c$, so $E$ verify $L_a^*$ in polynomial time.

# Problem 4 Numerous Purchase Coupons

1) We have $f(M, N) = \lg M + \lg N + 3N \lg M$, so $L = O(N \lg M + \lg N)$.

2) No, for a fixed $N$, we have $L = O(\lg M)$ while the time complexity of algorithm is $O(M) = O(2^L)$.

# Problem 5 Sorry, I am a NPC

1) We can sum up the value as $S$ in $O(n)$ time, simply let $T = S/2$ will do the reduction.

2) If we want $T$ in Q2, the remaining part wil be $S - T$, which means $T$ and $S - T$ are equivalent in Q2, so simply add a new item as $|2T - S|$, the ALGO1 will try to divide them into $(T, T)$ or $(S - T, S - T)$, since the new item will only be placed in one part, we finish the reduction in $O(n)$ time as previous problem.

3) If there are $K$ items which values are $(v_1, v_2, \ldots, v_K)$ in Q1, we can simply let $M = S/2, N = K, n_i = 1, b_i = s_i = v_i$, then check whether the "money" problem yield the best solution as $S/2$, which is a obviously polynomial reduction from Q1 to "money". By the previous problem, we also can reduct Q2 to Q1 to money in polynomial time.