# 补充题解 - 《经典》- 第 7 章 暴力求解法

## 习题7-7 埃及分数（Eg[y]ptian Fractions (HARD version), Rujia Liu' s Present 6, UVa12558）

## 习题7-8 数字谜（Digit Puzzle, ACM/ICPC Xi'an 2006, UVa12107）

## 习题7-9 立体八数码问题（ Cubic Eight-Puzzle , ACM/ICPC Japan 2006, UVa1604）

## 习题7-13 数字表达式（According to Bartjens, ACM/ICPC World Finals 2000, UVa817）

输入一个以等号结尾，前面只包含数字的表达式，插入一些加号、减号和乘号，使得运算结果等于2000。表达式里的整数不能有前导零（比如0100或者000都是非法的），运算符都是二元的（比如`2*-100*-10+0=`是非法的），并且符合通常的运算优先级法则。 输入数字个数不超过9。如果有多解，按照字典序从小到大输出；如果无解，输出IMPOSSIBLE。比如2100100=有三组解，按照字典序依次为`2*100*10+0=`、`2*100*10-0=`、`2100-100=`。

【分析】

依次遍历每个字符，除了第1个字符必须是某个数字的开头之外，某个字符d在表达式中有2种可能：

1. 继续附加到前面的非0数字之后。
2. 先插入一个运算符，然后将这个d作为一个新数字的开始，当然要求d必须非0。

所有字符决策完成之后，就形成了一个表达式，对其进行求值，符合条件的记录下来，排序输出即可，详情参考代码：

```cpp
using namespace std;
#define _for(i, a, b) for (int i = (a); i < (b); ++i)
#define _rep(i, a, b) for (int i = (a); i <= (b); ++i)

struct OP {
  int val;
  char op;
  OP(int v) : op(0) { val = v; }
  OP(char _op) : val(-1) { op = _op; }
};

ostream& operator<<(ostream& os, const OP& op) {
  if (op.op)
    os << op.op;
  else
    os << op.val;
  return os;
}

int eval(const vector<OP>& exp, int l, int r) {
  assert(l <= r);
  const auto& le = exp[l];
  if (l == r) {
    assert(le.op == 0 && le.op != -1);
    return le.val;
  }
  int ai = -1, mi = -1;
  _rep(i, l, r) {
    auto op = exp[i].op;
    if (op == '+' || op == '-') ai = i;
    if (op == '*') mi = i;
  }
  if (ai != -1) {
    int lval = eval(exp, l, ai - 1), rval = eval(exp, ai + 1, r);
    return exp[ai].op == '+' ? lval + rval : lval - rval;
  }
  assert(mi != -1);
  return eval(exp, l, mi - 1) * eval(exp, mi + 1, r);
}

const string OPS = "+-*";
char S[16];
int SZ;

void dfs(int cur, vector<OP>& exp, vector<string>& ans) {
  if (cur == 0) {
    exp.emplace_back((int)(S[0] - '0'));
    dfs(cur + 1, exp, ans);
    exp.pop_back();
```

```
    return;
  }
  if (cur == SZ) {
    if (eval(exp, 0, exp.size() - 1) == 2000) {
      stringstream ss;
      for (auto e : exp) ss << e;
      ans.push_back(ss.str());
    }
    return;
  }
  auto lv = exp.back().val;
  assert(lv != -1);
  if (lv != 0) {
    exp.back().val = lv * 10 + S[cur] - '0';
    dfs(cur + 1, exp, ans);
    exp.back().val = lv;
  }
  for (auto op : OPS) {
    exp.emplace_back(op), exp.emplace_back((int)(S[cur] - '0'));
    dfs(cur + 1, exp, ans);
    exp.pop_back(), exp.pop_back();
  }
}

int main() {
  vector<OP> exp;
  vector<string> ans;
  for (int t = 1; scanf("%s", S) == 1 && S[0] != '='; t++) {
    SZ = strlen(S) - 1, S[SZ] = 0;
    printf("Problem %d\n", t);
    if (strcmp(S, "2000") == 0) {
      puts("  IMPOSSIBLE");
      continue;
    }
    exp.clear(), ans.clear();
    dfs(0, exp, ans);
    sort(begin(ans), end(ans));
    if (ans.empty())
      puts("  IMPOSSIBLE");
    else
      for (auto& s : ans) printf("  %s=\n", s.c_str());
  }
}
```

## 习题7-14 小木棍（Sticks, ACM/ICPC CERC1995, UVa307）

乔治有一些同样长的小木棍，他把这些木棍随意地砍成几段，直到每段的长都不超过50。现在，他想把小木棍拼接成原来的样子，但是却忘记了自己最开始有多少根木棍和它们的长度。给出每段小木棍的长度，编程帮他找出原始木棍的最小可能长度。比如若砍完后有4根，长度分别为1, 2, 3, 4，则原来可能是2根长度为5的木棍，也可能是1根长度为10的木棍，其中5是最小可能长度。另一个例子是：砍之后的木棍有9根，长度分别为5, 2, 1, 5, 2, 1, 5, 2, 1，则最小可能长度为
6（5+1=5+1=5+1=2+2+2=6），而不是8（5+2+1）。

【分析】by 潘逸铭&陈锋

将输入的木棍按照长度从大到小排序之后，记其长度为A[1~N]，$S=\sum A_i$，原始木棍的长度L一定满足，S ≥ A1且S能被L整除，则不难想到从小到大遍历所有的L，然后依次尝试将所有的棍子组装到一起，如果某个L能够组装成功，则这个L就是所求的结果。对L进行判断的过程请参考如下回溯逻辑：

```
// len: 当前大棍子还差多少长度, num: 已经拼成的棍子数量, cur: 当前决策的小棍子
bool testLen(int len, int num, int cur, const int L) {
  if (num * L == S) return true;
  _for(i, cur, N) if (!used[i] && A[i] <= len) {
    used[i] = true;  // 尝试用第i个棍子
    if (len == A[i]) {
      if (testLen(L, num + 1, 0, L)) return true;
    } else {
      if (testLen(len - A[i], num, i + 1, L)) return true;
    }
    used[i] = false;                    // 发现用A[i]不行
    if (len == L && cur == 0) break;  // 作为第一节失败了，总不能把它剩下
    if (len == A[i]) break;             // 之后一定无法组装完成
    while (i + 1 < N && A[i + 1] == A[i]) i++;  // 相同长度的stick不要再尝试
  }
  return false;
}
```

这里考虑几个剪枝技巧:

1. 注意`testLen`函数中的`cur`参数，`cur`的作用是，在尝试拼接原先一根木棒的时候，不要从第一个木棒开始枚举，而从上一次放下的小木棒后一根开始枚举。当一根木棒被成功拼出来之后，拼接下一根的时候则要从第一根开始枚举。实际上就是从大到小考虑目标木棒中的每一根小木棒。
2. 将所有的棍子按照长度从大到小排序。先枚举长的小木棒能够比较快地拼出一根木棒，能减少解答树每个结点的分叉，剪枝越早越好。
3. 如果当前枚举的小木棒，是正在拼接的木棒的第一节，那么当拼接失败时，不要再枚举后面更短的小木棒。因为最终这个小木棒总要拼进去，不能剩下。
4. 如果当前枚举的小木棒i，是正在拼接的木棒的最后一节，那么当拼接失败时，也不需要再枚举后面更短的小木棒。如果后续可以拼装成功，那么在拼装成功的方案中i一定在某个组中，和之前拼装方案的事实矛盾。
5. 相同长度的木棒，不要多次尝试。

# 习题 7-16 找座位（Finding Seats Again, UVa11846）

有一个n×n(n<20)的座位矩阵里坐着k(k≤26)个研究小组。每个小组的座位都是矩形形状。输入每个小组组长的位置和该组的成员个数，找到一种可能的座位方案。下图是一组输入和对应的输出。

```
...4.2.   AAAABCC
...45..   DDDDBEF
222..3.   GHIIBEF
...2..3   GHJKBEF
.24...2   LLJKBMM
...2.3.   NOJPQQQ
22..3..   NOJPRRR
```

【分析】by 陈锋

不难想到一种决策方式：依次对每个小组的矩形大小以及位置进行决策，但是粗略估计一下，每个小组要考虑矩形的左上角x,y坐标，两个边长，即使每个决策都要考虑2种情况，那么总的状态空间差不多就是有16^K，肯定会超时。

考虑另外一种决策方法，从左到右，从上到下，一次考虑以当前格子为左上角的矩形的右下角坐标。这两个点的坐标就可以唯一确定一个矩形并且进行标记。如果当前格子已经被标记，则直接跳转到到下一个格子进行决策，所有格子决策完之后就代表成功完成了。遍历时要考虑几种如下几种剪枝策略：

1. 当前矩形的面积不能大于9(因为每个组成员个数都是1~9的数字)，否则直接退出。
2. 矩形中唯一组长的标号如果不能被某个边长整除，说明当前遍历的矩形边长无效，退出。
3. 矩形中不能出现多于两个组长，否则说明面积太大了，退出。
4. 矩形中不能出现已经被标记的格子，否则说明矩形和之前决策的格子冲突，退出。
5. 如果矩形中没有组长，或者唯一的组长所标识的数字小于矩形面积，则直接考虑下一种边长。

当前格子所在的矩形完成之后，就跳转到矩形的矩形的右上角所在格子的下一个格子进行决策。

```cpp
// UVa11846 Finding Seats Again
// 陈锋
using namespace std;
#define _for(i, a, b) for (int i = (a); i < (b); ++i)
#define _rep(i, a, b) for (int i = (a); i <= (b); ++i)
#define _zero(D) memset((D), 0, sizeof(D))
#define _init(D, v) memset((D), (v), sizeof(D))
#define _ri1(x) scanf("%d", &(x))
#define _ri2(x, y) scanf("%d%d", &(x), &(y))
#define _ri3(x, y, z) scanf("%d%d%d", &(x), &(y), &(z))
#define _ri4(a, b, c, d) scanf("%d%d%d%d", &(a), &(b), &(c), &(d))
typedef long long LL;

const int MAXN = 20;
int N, K;
char G[MAXN][MAXN], Ans[MAXN][MAXN];
bool dfs(const char c, int pos) { // 当前决策的格子编号，以及所要放置的字符
  while(pos != N*N && Ans[pos/N][pos%N] != '.') pos++; // 忽略已经决策过的格子
  if(pos == N*N) return true;
```

```
  int x1 = pos%N, y1 = pos/N; // uppper-left corner
  _for(y2, y1, N) _for(x2, x1, N){ // bottom-right corner
    int lcnt = 0, area = (y2 - y1 + 1) * (x2 - x1 + 1); // leader count, area
    if(area > 9 || Ans[y2][x2] != '.') break; // max area ≤ 9
    int ld = -1; // leader digit
    bool used = false, too_large = false;
    _rep(y, y1, y2) { // every cell
      _rep(x, x1, x2){
        char gc = G[y][x];
        if(isdigit(gc)) {
          lcnt++, ld = gc - '0';
          // too many leaders, area too large, width invalid
          if(lcnt > 1 || area > ld || ld%(y2-y1+1)) {
            too_large = true;
            break;
          }
        }
        if(Ans[y][x] != '.') {
          used = true;
          break;
        }
      }
      if(used || too_large) break;
    }
    if(used || too_large) break;
    if(lcnt == 0 || ld > area) continue; // area too small, continue trying
    _rep(y, y1, y2) _rep(x, x1, x2) Ans[y][x] = c; // fill the region
    if(dfs(c+1, pos+(x2-x1+1))) return true; // go to right of the rectangle
    _rep(y, y1, y2) _rep(x, x1, x2) Ans[y][x] = '.';
  }
  return false;
}

int main() {
  while(_ri2(N, K) == 2 && N && K){
    _init(Ans, '.');
    _for(i, 0, N) _for(j, 0, N) scanf(" %c ", &G[i][j]);
    dfs('A', 0);
    _for(i, 0, N) {
      _for(j, 0, N) putchar(Ans[i][j]);
      puts("");
    }
  }
  return 0;
}
```

```
65 // 20432842 11846 Finding Seats Again Accepted  C++11 0.000 2017-12-02
   11:46:07
```

# 习题 7-17 Gokigen Naname 谜题（ Gokigen Naname, UVa11694）

首先考虑建模，将每个交叉点看做图的一个结点，所有的斜线就是边。根据每个交叉点上的数字其实都是一种线索。采用类似于玩数独游戏思路，对于结点v(i,j)，记其上标记的数字为d, 周围方块中还未填上斜线的格子数目为f，已经填上和v连接的斜线的格子数目为c。分别考虑以下情况：

1. c = d且f > 0，说明v的连接数目已经满足并且周围还有空的格子。需要将空的格子中全部填成不与v连接的斜线。
2. c + f = d，说明剩下的空格子需要全部填成与v连接的斜线。

不停的轮询所有v，满足以上条件的就进行填充。当没有可以填充的v并且还剩下格子未填充时。就需要采用回溯法，对每个格子的斜线方向进行决策，在任何格子填入斜线连接两个v之前要判断其连通性，如果已经连通则不能填入。

本题实现代码细节较多，请参考代码。