Livestock Lineup

Josh-Cena

2020 年 12 月 2 日

Every day, Farmer John milks his 8 dairy cows, named Bessie, Buttercup, Belinda, Beatrice, Bella, Blue, Betsy, and Sue.

The cows are rather picky, unfortunately, and require that Farmer John milks them in an order that respects N constraints. Each constraint is of the form "X must be milked beside Y", stipulating that cow X must appear in the milking order either directly after cow Y or directly before cow Y.

Please help Farmer John determine an ordering of his cows that satisfies all of these required constraints. It is guaranteed that an ordering is always possible. If several orderings work, then please output the one that is alphabetically first. That is, the first cow should have the alphabetically lowest name of all possible cows that could appear first in any valid ordering. Among all orderings starting with this same alphabetically-first cow, the second cow should be alphabetically lowest among all possible valid orderings, and so on.

数据规模	内存限制	运行时间
$1 \leq N \leq 7$	256 MB	2.0 s

题解. 这道题在理解了题目的需求——生成一个符合给定约束的字典序最小的排列后，应该难度不高。我们可以按字典序生成全部的排列（一共有 $8! = 40320$ 种），然后输出第一个满足所有约束条件的。如果不会用回溯算法生成全排列，可能需要借助 `algorithm` 中的 `next_permutation` 函数。这也是典型的铜组思路：因为规模极小，可以暴力枚举之后挑选解而不是构造解。

暴力法的代码：

```
/**
 * Adopted from official solution at
 * http://www.usaco.org/current/data/sol_lineup_bronze_dec19.html
 */
#include <iostream>
#include <fstream>
#include <vector>
#include <algorithm>

using namespace std;

string names[8] = {"Beatrice", "Belinda", "Bella", "Bessie", "Betsy", "Blue",
    "Buttercup", "Sue"};
// beside_a 和 beside_b 中对应下标的奶牛表示一组约束关系
vector<string> beside_a, beside_b;
int n;

int getID(string name) {
   for (int i = 0; i < 8; i++)
      if (names[i] == name)
         return i;
   return -1;
}

bool satisfies_constraints() {
   for (int i = 0; i < n; i++)
      if (abs(getID(beside_a[i]) - getID(beside_b[i])) != 1)
         return false;
   return true;
}

int main() {
   ifstream fin("lineup.in");
   ofstream fout("lineup.out");
   fin >> n;
   string a, b;
   for (int i = 0; i < n; i++) {
      fin >> a >> b >> b >> b >> b >> b;
      beside_a.push_back(a);
      beside_b.push_back(b);
   }
```

```
    // 遍历所有的8头奶牛的排列，输出第一个满足约束的解
    do {
        if (satisfies_constraints()) {
            for (int i = 0; i < 8; i++)
                fout << names[i] << endl;
            return 0;
        }
    } while (next_permutation(names.begin(), names.end()));
    return 0;
}
```

当然，这种方法对于有一些竞赛经验的参赛者来说反倒不容易想到。这些参赛者会试图通过约束条件来构造解。这需要一点点贪心的思想：为了让排列字典序最小，就一定要让每一位上的奶牛字典序尽可能小。我们可以把一个排列看作一个“约束链”，其中每一头奶牛都因为它相邻位置奶牛的约束而只有唯一的可能。每完成一条约束链的连接后，都可以从剩下未被安排进队的奶牛中挑选编号最小的，但不能是有两个未满足的约束的（因为一个“约束链”中头和尾的奶牛都只能和它相邻的一头奶牛间有约束），然后在确定了链头之后，就可以非常自然地得到整个链条。重复同样的构造过程，直到所有奶牛都被添加入队列。

构造法的代码：

```
#include <iostream>
#include <fstream>

using namespace std;

struct cow {
    int adj[2]; // 需要和这头奶牛相邻的奶牛的ID
    int adjcnt; // 这头奶牛一共有几个约束条件；决定了能否把它放在约束链的开头
    bool chosen; // 是否已经进队
} cows[8];
string names[8] = {"Beatrice", "Belinda", "Bella", "Bessie", "Betsy", "Blue",
    "Buttercup", "Sue"};

int getID(string name) {
    for(int i = 0; i < 8; i++)
        if(names[i] == name)
            return i;
    return -1;
}
```

```

int main() {
   ifstream fin("lineup.in");
   ofstream fout("lineup.out");
   int n;
   fin >> n;
   string a, b;
   for (int i = 0; i < n; i++) {
      fin >> a >> b >> b >> b >> b >> b;
      cows[getID(a)].adj[cows[getID(a)].adjcnt++] = getID(b);
      cows[getID(b)].adj[cows[getID(b)].adjcnt++] = getID(a);
   }
   int prev = -1;
   // 每次循环向队列中添加一头奶牛; 如果上一头奶牛没有更多的约束条件了, 则可以选择一头新的,
       否则选择需要和上一头相邻的奶牛
   for (int _ = 0; _ < 8; _++) {
      if (_ == 0 || cows[prev].adjcnt == 0) {
         for (int i = 0; i < 8; i++) {
            if (!cows[i].chosen && cows[i].adjcnt < 2) {
               cows[i].chosen = true;
               fout << names[i] << endl;
               prev = i;
               break;
            }
         }
      } else if (cows[prev].adjcnt == 1) {
         int i = cows[prev].adj[0];
         cows[i].chosen = true;
         // 这里的操作是在把cows[i]添加入队列的同时 “删除” 掉它已经满足的那条约束
         cows[i].adjcnt--;
         if(cows[i].adj[0] == prev)
            cows[i].adj[0] = cows[i].adj[1];
         fout << names[i] << endl;
         prev = i;
      }
   }
   return 0;
}
```