Aufgabe 1 - Blumenbeet

Team Mineraldifferenz (00694)

Fynn Kiwitt, Erik Klein, Adrian Krauss

Lösungsidee

Das Problem lässt sich mit Brute-Force lösen. Es gibt 9 Felder, die jeweils maximal 7 Farben annehmen können und somit 7^9 verschiedene Bepflanzungen. Diese kann man rekursiv konstruieren.

Alle Bepflanzungen werden durchgegangen und ihre Punktzahl berechnet. Dafür werden jeweils alle 16 Nachbarschaftsbeziehungen der Felder durchgegangen und geprüft, ob sie einem Wunsch entsprechen. Die Bepflanzung mit der größten Punktsumme wird ausgegeben.

Das dauert bis zu 10s. Für 2 Fälle haben wir deshalb schnellere Lösungen gefunden:

- Es gibt nur Wünsche der Form A A, d.h. die aus gleichen Farben bestehen. Dann werden alle Felder mit der Farbe des Wunsches mit der größten Punktzahl P bepflanzt. Die Punktsumme ist 16*P.
- Es gibt nur Wünsche der Form A B, d.h. die aus unterschiedlichen Farben bestehen. Die Farben A B des Wunsches mit der größten Punktzahl P werden gewählt und so angeordnet:

  A

  B B

  A A A

  B B

  A

  Die Punktsumme ist 12*P.

Anfangs wird geprüft, ob einer dieser Fälle eintritt und wenn ja die schnellere Lösung verwendet. Das dauert nur 0.04s, und spart in den meisten Fällen, so auch in allen BwInf-Beispielen, enorm Zeit.

Wenn es Wünsche beider Formen gibt, wird die Brute-Force Lösung verwendet.

Umsetzung

Die Idee wurde in C++ umgesetzt. Die Farben werden mit Integern repräsentiert, wofür zwei Maps verwendet werden: 1. von string zu int, 2. von int zu string. Die Wünsche und Bepflanzungen werden in Integer-Arrays gespeichert.

Beim Einlesen werden die boolschen Variablen AA und AB gesetzt, die danach angeben, ob es Wünsche der Form A A bzw. A B gibt. Darauf folgt die oben beschriebene Fallunterscheidung durch if-Abfragen.

Die schnelleren Lösungen sind trivial, zu erklären ist noch die Brute-Force Lösung:

Die Funktion gen(i) implementiert die rekursive Konstruktion der Bepflanzung und verwendet Backtracking. Anfangs wird gen(0) aufgerufen. i ist der Index des Feldes, dessen Farbe als nächstes gesetzt wird. col[i] ist entweder schon richtig, sodass gen(i+1) aufgerufen wird, oder noch zu niedrig. Dann wird col[i] inkrementiert und wieder gen(i) aufgerufen. Danach muss col[i] wieder dekrementiert werden. So werden nacheinander alle Bepflanzungen konstruiert. Eine Bepflanzung ist fertig, wenn i = 9, was den Basisfall der Rekursion darstellt. In diesem Fall wird die Punktsumme bestimmt. Alle Nachbarschaftsbeziehungen der Felder werden in einer for-Schleife durchgegangen und deren Punktzahl bestimmt.

Das geschieht mit der Funktion scr(i,j), die alle Wünsche durchgeht und einen mit den Farben i und j sucht, in welchem Fall dessen Punktzahl zurückgegeben wird. Gibt es keinen passenden Wunsch, wird 0 zurückgegeben.

Ist die Punktsumme größer als die der besten bisher gefundenen Lösung, wird diese beste Lösung durch die derzeitige Lösung ersetzt. So hat man nach dem Durchgehen von allen Lösungen die Lösung mit der absolut größten Punktsumme gefunden.

Beispiele

Die Nummern der BwInf-Website:

blumen.txt:

Bewertung: 36

Hochbeet: P1:rot, P2:blau, P3:blau, P4:rot, P5:rot, P6:rot, P7:blau, P8:blau, P9:rot.

blumen1.txt:

Bewertung: 36

Hochbeet: P1:rot, P2:tuerkis, P3:tuerkis, P4:rot, P5:rot, P6:rot, P7:tuerkis, P8:tuerkis, P9:rot.

blumen2.txt:

Bewertung: 36

Hochbeet: P1:tuerkis, P2:rot, P3:rot, P4:tuerkis, P5:tuerkis, P6:tuerkis, P7:rot, P8:rot, P9:tuerkis.

blumen3.txt:

Bewertung: 36

Hochbeet: P1:rot, P2:tuerkis, P3:tuerkis, P4:rot, P5:rot, P6:rot, P7:tuerkis, P8:tuerkis, P9:rot.

blumen4.txt:

Bewertung: 36

Hochbeet: P1:rot, P2:tuerkis, P3:tuerkis, P4:rot, P5:rot, P6:rot, P7:tuerkis, P8:tuerkis, P9:rot.

blumen5.txt:

Bewertung: 36

Hochbeet: P1:rot, P2:tuerkis, P3:tuerkis, P4:rot, P5:rot, P6:rot, P7:tuerkis, P8:tuerkis, P9:rot.

eigenes_beispiel.txt:

7

3

rot rot 1

blau blau 2

orange orange 3

Bewertung: 48

Hochbeet: P1:orange, P2:orange, P3:orange, P4:orange, P5:orange, P6:orange, P7:orange, P8:orange, P9:orange.

eigenes_beispiel1.txt:

7

3

gruen blau 3

rot rot 2

gruen gruen 1

Bewertung: 38

Hochbeet: P1:gruen, P2:gruen, P3:blau, P4:blau, P5:gruen, P6:gruen, P7:gruen, P8:blau, P9:gruen.

eigenes_beispiel2.txt:

7

3

rot rosa 2

rosa rosa 2

rot rot 1

Bewertung: 32

Hochbeet: P1:rot, P2:rosa, P3:rosa, P4:rot, P5:rosa, P6:rot, P7:rosa, P8:rosa, P9:rot.

eigenes_beispiel3.txt:

7

5

rot blau 1

gruen blau 2

blau schwarz 2

blau blau 1

gelb rot 2

Bewertung: 27

Hochbeet: P1:blau, P2:blau, P3:gruen, P4:gruen, P5:blau, P6:blau, P7:blau, P8:gruen, P9:blau.

eigenes_beispiel4.txt:

1 1

rot rot 0

Bewertung: 0

Hochbeet: P1:rot, P2:rot, P3:rot, P4:rot, P5:rot, P6:rot, P7:rot, P8:rot, P9:rot.

Quellcode

```
#include <bits/stdc++.h>
#define mp make_pair
using namespace std;

int F, W, maxsc;
int col[9], maxcol[9];
int wish[7][3]; // i, j, score

int MXAA_sc, MXAA_A, MXAB_sc, MXAB_A, MXAB_B;
bool AA, AB;

map<string,int> id;
map<int,string> bac;

int scr(int i, int j) {
    for(int k = 0; k < W; k++)
        if(((i == wish[k][0]) && (j == wish[k][1]))
        || ((i == wish[k][1]) && (j == wish[k][0]))) return wish[k][2];
    return 0;
}

void gen(int i) {
    if(i == 9) {
        // process sol
        // calc score
        int sc = 0;
        for(pair<int,int> x : { mp(0,1),mp(0,2),mp(1,2),mp(1,3),
                                mp(1,4),mp(2,4),mp(2,5),mp(3,4),
                                mp(3,6),mp(4,5),mp(4,6),mp(4,7),
                                mp(5,7),mp(6,7),mp(6,8),mp(7,8)})
            sc += scr(col[x.first],col[x.second]);
        if(maxsc < sc) {
            maxsc = sc;
            for(int j = 0; j < 9; j++) maxcol[j] = col[j];
        }
        return;
    }
    gen(i+1); // leave col[i]
    if(col[i]+1 < W) {
        col[i]++;
        gen(i);
        col[i]--;
    }
}
```

```
signed main() {
    cin >> F >> W;
    string s1, s2;
    for(int k = 0, s; k < W; k++) {
        cin >> s1 >> s2 >> wish[k][2];
        if(id.find(s1) == id.end()) {
            s = id.size();
            id[s1] = s, bac[s] = s1;
        }
        if(id.find(s2) == id.end()) {
            s = id.size();
            id[s2] = s, bac[s] = s2;
        }
        wish[k][0] = id[s1];
        wish[k][1] = id[s2];
    }
    for(int i = 0; i < W; i++) {
        if(wish[i][0] == wish[i][1]) {
            AA = 1;
            // find best A A
            if(wish[i][2] > MXAA_sc) {
                MXAA_sc = wish[i][2];
                MXAA_A  = wish[i][0];
            }
        } else  {
            AB = 1;
            // find best A B
            if(wish[i][2] > MXAB_sc) {
                MXAB_sc = wish[i][2];
                MXAB_A  = wish[i][0];
                MXAB_B  = wish[i][1];
            }
        }
    }

    if(AA && AB) {
        gen(0);
        cout << "Bewertung: " << maxsc << "\n";
        cout << "Hochbeet: ";
        for(int i = 0; i < 8; i++) cout << 'P' << i+1 << ':' << bac[maxcol[i]] << ", ";
        cout << "P9:" << bac[maxcol[8]] << ".\n";
    } else if(AA) {
        cout << "Bewertung: " << 16*MXAA_sc << "\n";
        cout << "Hochbeet: ";
        for(int i = 0; i < 8; i++) cout << 'P' << i+1 << ':' << bac[MXAA_A] << ", ";
        cout << "P9:" << bac[MXAA_A] << ".\n";
    } else if(AB) {
        cout << "Bewertung: " << 12*MXAB_sc << "\nHochbeet";
        cout << ": P1:" << bac[MXAB_A]                                                          // A
             << ", P2:" << bac[MXAB_B] << ", P3:" << bac[MXAB_B]                                // B
             << ", P4:" << bac[MXAB_A] << ", P5:" << bac[MXAB_A] << ", P6:" << bac[MXAB_A]      // A
             << ", P7:" << bac[MXAB_B] << ", P8:" << bac[MXAB_B]                                // B
             << ", P9:" << bac[MXAB_A]                                                          // A
             << ".\n";
    }
    return 0;
}
```