Aufgabe 1: Wörter aufräumen

00759 - TranshumanTechnocrator - Erik Klein

1 Aufgabe

Gegeben ist eine Liste von Start-Wörtern, mit der ein Lückentext vervollständigt werden muss. Dieser ist widerum eine Sequenz aus Ziel-Wörtern, bei denen aber manche Buchstaben eine Lücke sind, d.h. beliebige Buchstaben an diese Stelle passen. Die Frage ist, welche s-Wörter für welche z-Wörter eingesetzt werden sollen, sodass sie übereinstimmen.

2 Lösungsidee

Die Aufgabe wurde mit Backtracking gelöst.

Um zu prüfen, ob ein s-Wort an eine z-Stelle passt, wird geprüft, ob das dortige z-Wort dieselbe Länge hat, und ob alle sich entsprechenden Buchstaben entweder gleich, oder beim z eine Lücke sind.

Erst wird eine boolsche 2D-Tabelle berechnet, die speichert, ob das i-te s-Wort an die j-te z-Stelle passt. Dies dauert $\mathcal{O}(n \cdot m \cdot l)$, wobei es n s-Wörter und m z-Wörter[1] gibt und die Maximallänge eines Worts l beträgt. In Praxis verläuft das viel schneller, weil bei vielen Überprüfungen schon die Wortlängen nicht übereinstimmen, und nicht alle Wörter Maximallänge haben.

Dann beginnt die rekursive Suche nach einer Lösung mit $i = 0$. Wenn die Rekursion mit Parameter i aufgerufen wird, sollen für die Stellen $0...i-1$ bereits passende s-Wörter gefunden worden sein. In einer boolschen Reihung wird dabei gespeichert, welche s-Wörter in der derzeitigen Teillösung verwendet werden. Ihre Reihenfolge wird dabei mit einem Stack gespeichert. Dann geht die Funktion alle s-Wörter durch, die noch nicht verwendet worden sind, setzt sie jeweils an die i-te z-Stelle, wenn das möglich ist (das wird in der Tabelle nachgeschaut), und ruft $rekursion(i+1)$ auf. Ist die Rekursion bei $i = m$ angekommen, wurde eine Lösung gefunden, und die Suche kann abgebrochen werden. Die Suche dauert $\mathcal{O}(n^m)$. Findet sie keine Lösung, ist das Rätsel unlösbar.

3 Umsetzung

Das Verfahren wurde in C++ implementiert. Die Funktion $pos()$ berechnet wie beschrieben, ob eine das s-Wort an der i-ten Stelle an die j-te z-Stelle passt. Die 2D-Tabelle ist $poss$. Die rekursive Suchfunktion habe ich dfs genannt. Zuerst wird die Eingabe eingelesen, wobei für die spätere Ausgabe außerdem eine Reihung aft verwaltet wird, die speichert, nach welchem Wort welches Satzzeichen folgt. Der Stack, um die s-Wörter der Teillösung in Reihenfolge zu speichen, heißt st, und die boolsche Reihung, die speichert, ob ein s-Wort in der Teillösung verwendet wurde, heißt in.

4 Quellcode

```
#include <bits/stdc++.h>
using namespace std;
enum
{
    // einstellen: fuer so viele
    // woerter funktioniert die
    // loesung maximal
    MAXN = 1005,
    MAXM = 1005,
};
int N, M;
```

[1] Damit überhaupt alle z-Stellen ausgefüllt werden können, muss $n \geq m$.

```
bool in[MAXN]; // speichert welche woerter verwendet
char aft[MAXN];// speichert satzzeichen nach index
string s[MAXN], z[MAXM]; // woerter
stack<int> st; // speichert reihenfolge teilloesung

bool poss[MAXN][MAXM];
bool pos(int i, int j) {
    // laengen verschieden
    if(s[i].length() != z[j].length()) return 0;

    for(int k = 0; k < s[i].length(); k++)
        // sich entsprechende buchstaben verschieden
        if(z[j][k] != '_' && s[i][k] != z[j][k]) return 0;

    // passt
    return 1;
}

void dfs(int i) {
    if(i == M) {
        // loesung gefunden
        vector<int> sol(M);
        for(int j = M-1; j >= 0; j--) {
            // oberstes element des stacks
            // entspricht letztem wort der
            // loesung da zuletzt hinzugefuegt
            sol[j] = st.top(); st.pop();
        }
        // loesung ausgeben
        for(int j = 0; j < M; j++) {
            cout << s[sol[j]];
            // satzzeichen wieder einfuegen
            if(aft[j]) cout << aft[j];
            cout << '␣';
        }
        cout << '\n';
        // suche abbrechen
        exit(0);
    }
    for(int j = 0; j < N; j++)
        if(!in[j] && poss[j][i]) {
            // j-tes s-Wort an
            // i-te z-Stelle

            in[j] = 1; // als verwendet markieren
            st.push(j);// reihenfolge in
                       // stack speichern

            // rekursion aufrufen
            dfs(i+1);

            // rueckgaengig machen
            st.pop();
            in[j] = 0;
        }
}

signed main()
{
    ios::sync_with_stdio(0);
    {
        // einlesen
        string c;
```

```
75          // zeile 1: z-Woerter
            getline(cin,c);
77          for(int i = 0; i < c.length(); i++) {
                if(isalpha(c[i]) || c[i] == '_') z[M] += c[i];
79              else if(c[i] == '␣') M++;
                else aft[M] = c[i]; // satzzeichen
81          }
            M++;
83          // zeile 2: s-Woerter
            getline(cin,c);
85          for(int i = 0; i < c.length(); i++) {
                if(c[i] == '␣') N++;
87              else    s[N] += c[i];
            }
89          N++;
        }
91
        // tabelle berechnen
93      for(int i = 0; i < N; i++)
            for(int j = 0; j < M; j++)
95              poss[i][j] = pos(i,j);

97      dfs(0); // suche starten

99      // wenn hier angekommen, hat
        // suche keine loesung gefunden
101     cout << "unloesbar\n";
        return 0;
103 }
```

main.cpp

5 Beispiele

Hier zuerst die Ausgaben für die Beispiellösungen aus der Aufgabenstellung:
0. oh je, was für eine arbeit!
1. Am Anfang wurde das Universum erschaffen. Das machte viele Leute sehr wütend und wurde allenthalben als Schritt in die falsche Richtung angesehen.
2. Als Gregor Samsa eines Morgens aus unruhigen Träumen erwachte, fand er sich in seinem Bett zu einem ungeheueren Ungeziefer verwandelt.
3. Informatik ist die Wissenschaft von der systematischen Darstellung, Speicherung, Verarbeitung und Übertragung von Informationen, besonders der automatischen Verarbeitung mit Digitalrechnern.
4. Opa Jürgen blättert in einer Zeitschrift aus der Apotheke und findet ein Rätsel. Es ist eine Liste von Wörtern gegeben, die in die richtige Reihenfolge gebracht werden sollen, so dass sie eine lustige Geschichte ergeben. Leerzeichen und Satzzeichen sowie einige Buchstaben sind schon vorgegeben.

Nun zwei Zitate von Obama:

The be_ _ _ _ to n_t fe_l _opele_s i_ to ge_ up an_ _o _omething. _o_t _a_t _or g_ _d _h_ng_ to happe_ _ _ _ou. If _ _ _ _ _ o_ _ a_d mak_ so_ _ goo_ t_ _ _g_ _ _ _ _ _ _, y_u wil_ _ill _he wo_ld _i_h _o_e, y_u _ill f_ _ _ _ _ursel_ _ _t_ hop_.
hope with yourself fill will you hope with world the fill will you happen things good some make and out go you If you to happen to things good for wait Dont something do and up get to is hopeless feel not to way best The
Ausgabe:
The best you to not feel hopeless is to get up and go something. Dont wait for good things to happen do you. If way to out and make some good things happen, you will fill the world with hope, you will fill yourself with hope.
Hier gab es offensichtlich mehrere Lösungen. Z.B. go, do und to wurden vertauscht.

Wh_ _ _v_ real_ _e_ i_ _ _ _ _ li_ _ do_ _ _t _ount fo_ _u_h un_ _s_ you_e wil_ _ _ _ to d_ you_ _m_ll par_ to l_ave o_r c_il_r_n a_ _ of o_ _ c_ _ldr_n a _ _ _ _e_ _ _r_d. Any fo_l _ _n

h_ve a _hi__. _h_t __e__t _a_e _ou _ _ather. I__ the _o_r_g_ __ r_i__ a chi__ ___t m_ke_ you _ fath__.
father a you makes that child a raise to courage the Its father a you make doesnt That child a have can fool Any world better a children our of all children our leave to part small your do to willing youre unless much for count doesnt life that is realized Ive What
Ausgabe:
What Ive realized is that life doesnt count for much unless youre willing to do your small part to leave our children all of our children a better world. Any fool can have a child. That doesnt make you a father. Its the courage to raise a child that makes you a father.

Ich habe leider nur wenige Beispiele, da es in Latex sehr umständlich ist, Unterstriche _ einzufügen. Hier noch ein paar Informatik-Zitate:

Comp_t__ Sc_en_e _s _o ___e a__ut computer_ _han ast_____ _s _bou_ _eles_o__s.
telescopes about is astronomy than computers about more no is Science Computer
Ausgabe:
Computer Science is no more about computers than astronomy is about telescopes.

oftwar dev_lo___n_ h_s be_n, i_, a_d ___l _i_e_y __ma_n f_n_a____ally ____. _uil__ng _u__i_y s__te_s i_volve_ a_ e_s_n_ia_ _nd ir____cib_e _omp_e_i_y, _h_ch is ___ t_e enti_e histor_ of s_ftw__e e_ginee__ng _a_ be __aracteri__d __ o__ o_ _isi_g leve__ _f _b_t_____. As such, t_e tas_ __ th_ _oftwa__ d___lo__en_ te_m is to enginee_ the _llu_ion of s_mplic___. N_ne_h__ess, soft__rei_t_n_iv_ sys_ems can a_p__fy h____ _ntell__e_c_, _ut t_ey __n__t r_pl__e hu_an judg_e_t; __f_w__eint_nsive _ys_e_s can _use, coor_in__e, cla__if_, an_ ___l___ info_mation, but __ey canno_ c_eate _n__le___. In o_he_ w_rd_, n__ ____y_h___ w_ w_nt t_ b_i_d can __ buil_: __er_ _xist pr_gm_t_c t__ore_ical and _ec_____l l_mits __a_ _a__ so_t__r_ develpm_nt h_r_ i_ no_ i_p_ssi_l_. Furt_erm___, _ot eve_y_hin_ we wa__ t_ bui_d s_ou_d be b___t: __ere ___s_ __r_l ec_no__c, __c___, _nd politi_al _im_t_ th_t g_ve__ human i_____ry. _rom fun_am_n_al _o _u___, t_ese _r_ ___ f__tors tht defi_e _he _imits __ s__tw_re, f_ct_rs that se___ate _ur v_s_on __om _x___t_on.
execution from vision our separate that factors software of limits the define tht factors the are these human to fundamental From industry human govern that limits political and social economic moral exist there built be should build to want we everything not Furthermore impossible not if hard develpment software make that limits technical and theoretical pragmatic exist there built be can build to want we everything not words other In knowledge create cannot they but information analyze and classify coordinate fuse can systems softwareintensive judgment human replace cannot they but intelligence human amplify can systems softwareintensive Nonetheless simplicity of illusion the engineer to is team development software the of task the such As abstraction of levels rising of one as characterized be can engineering software of history entire the why is which complexity irreducible and essential an involves systems quality Building hard fundamentally remain likely will and is been has development Software
Ausgabe:
Software development has been, is, and will likely remain fundamentally hard. Building quality systems involves an essential and irreducible complexity, which is why the entire history of software engineering can be characterized as one of rising levels of abstraction. As such, the task of the software development team is to engineer the illusion of simplicity. Nonetheless, software-intensive systems can amplify human intelligence, but they cannot replace human judgment; software-intensive systems can fuse, co-ordinate, classify, and analyze information, but they cannot create knowledge. In other words, not everything we want to build can be built: there exist pragmatic theoretical and technical limits that make software development hard if not impossible. Furthermore, not everything we want to build should be built: there exist moral economic, social, and political limits that govern human industry. From fundamental to human, these are the factors tht define the limits of software, factors that separate our vision from execution.