

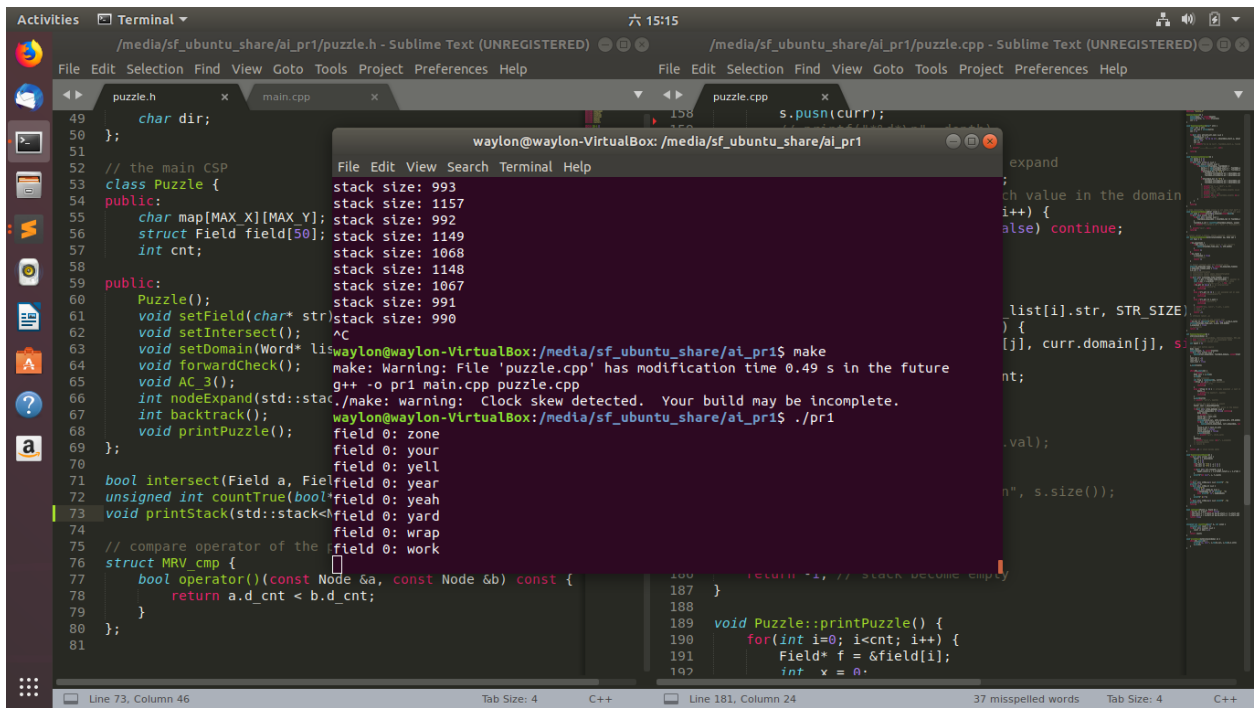
AI Project 1 Report

0516076 施威綸

Tests

- Brute force (node consistent)

I did not know how long would it take to finish.



The screenshot shows a Sublime Text editor with two files open: `puzzle.h` and `puzzle.cpp`. The `puzzle.h` file contains the definition of the `Puzzle` class, including a `map` of size `MAX_X` by `MAX_Y`, a `Field` struct, and a `cnt` variable. The `puzzle.cpp` file contains the implementation of the `Puzzle` class, including methods for setting fields, checking intersections, and printing the puzzle. A terminal window is open in the foreground, showing the command `make` being executed, which produces a warning about a clock skew and then runs the program. The output of the program shows the puzzle state: `field 0: zone`, `field 0: your`, `field 0: yell`, `field 0: year`, `field 0: yeah`, `field 0: yard`, `field 0: wrap`, and `field 0: work`.

The time complexity is $O(b^n)$, where the branching factor here is 500 approximately, the time to complete searching is all by luck.

So I created a smaller word list (domain) to ensure the code works properly.

- MRV/LCV

The program consumes more time in most cases in this method.

More nodes are visited.

```
#define MAX_FIELD 50
#define STR_SIZE 20

extern struct Word word_list;

// struct Line {
//   int start_x;
//   int start_y;
//   int length;
//   char dir;
// };

// for word list only
struct Word {
  char str[STR_SIZE];
  unsigned int len;
};

struct Node {
  int fid; // field id (variable)
  int wid; // word id of
  char val[STR_SIZE];
  bool domain[MAX_FIELD][MAX_FIELD];
  unsigned int d_cnt[MAX_FIELD][MAX_FIELD];
  bool root;
  bool expanded[MAX_FIELD][MAX_FIELD];
};

struct Field {
  int id; // variable
  char val[STR_SIZE];
  char* itrsc[STR_SIZE];
  bool domain[MAX_DOMAIN][MAX_DOMAIN];
  unsigned int d_cnt;
```

```
waylon@waylon-VirtualBox:/media/sf_ubuntu_share/ai_pr1$ ./pr1 | grep field | wc -l
204
waylon@waylon-VirtualBox:/media/sf_ubuntu_share/ai_pr1$
```

```
c i t y
r o w e
w i n d
```

visited nodes

- **AC-3 & forward checking**

Not yet finished.

I had tried AC-3 before searching, but failed.

I have not figured out when to update the neighbors, which is the main cause of failure.

Discussion

At first, I did not realize that each node stands for a state. So I encountered a lot of difficulties.

The 4th puzzle are not solved in the end, while the program had run more than an hour.

The depth of the last puzzle is 12. Compare to the previous ones (5~6), the time may be 500^6 (up) longer.

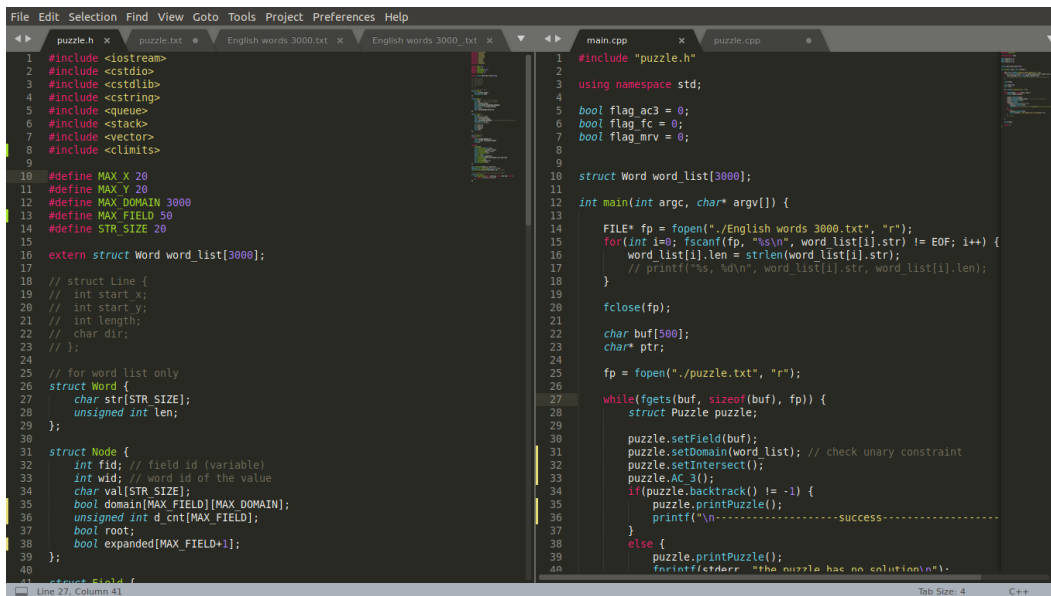
- **Observation**

Forward checking and arc consistency will reduce the branching factor and may be critical to improve performance. (number of values in domain has reduced)

- **Remaining problem**

To finish forward checking.

Appendix



```
1 #include <iostream>
2 #include <cstdio>
3 #include <cstdlib>
4 #include <cstring>
5 #include <queue>
6 #include <stack>
7 #include <vector>
8 #include <limits>
9
10 #define MAX_X 20
11 #define MAX_Y 20
12 #define MAX_DOMAIN 3000
13 #define MAX_FIELD 50
14 #define STR_SIZE 20
15
16 extern struct Word word_list[3000];
17
18 // struct Line {
19 //   int start_x;
20 //   int start_y;
21 //   int length;
22 //   char dir;
23 // };
24
25 // for word list only
26 struct Word {
27   char str[STR_SIZE];
28   unsigned int len;
29 };
30
31 struct Node {
32   int fid; // field id (variable)
33   int wid; // word id of the value
34   char val[STR_SIZE];
35   bool domain[MAX_FIELD][MAX_DOMAIN];
36   unsigned int d_cnt[MAX_FIELD];
37   bool root;
38   bool expanded[MAX_FIELD+1];
39 };
40
41 #include "puzzle.h"
42 using namespace std;
43
44 bool flag_ac3 = 0;
45 bool flag_fc = 0;
46 bool flag_mrv = 0;
47
48 struct Word word_list[3000];
49
50 int main(int argc, char* argv[]) {
51
52   FILE* fp = fopen("./English words 3000.txt", "r");
53   for(int i=0; fscanf(fp, "%s\n", word_list[i].str) != EOF; i++) {
54     word_list[i].len = strlen(word_list[i].str);
55     // printf("%s, %d\n", word_list[i].str, word_list[i].len);
56   }
57   fclose(fp);
58
59   char buf[500];
60   char* ptr;
61
62   fp = fopen("./puzzle.txt", "r");
63   while(fgets(buf, sizeof(buf), fp)) {
64     struct Puzzle puzzle;
65
66     puzzle.setField(buf);
67     puzzle.setDomain(word_list); // check unary constraint
68     puzzle.setIntersect();
69     puzzle.AC_3();
70     if(puzzle.backtrack() != -1) {
71       puzzle.printPuzzle();
72       printf("\n-----success-----\n");
73     }
74     else {
75       puzzle.printPuzzle();
76       fprintf(stderr, "this puzzle has no solution\n");
77     }
78   }
79 }
```

main.c + puzzle.h (1)

```
File Edit Selection Find View Goto Tools Project Preferences Help
puzzle.h x puzzle.txt x English words 3000.txt x English words 3000_txt x
40 struct Field {
41     int id; // variable
42     char val[STR_SIZE];
43     char* itrsc[STR_SIZE];
44     bool domain[MAX_DOMAIN]; // trimmed domain before search
45     unsigned int d_cnt;
46     // struct Line line;
47     int start x;
48     int start y;
49     int length;
50     char dir;
51 };
52
53 // the main CSP
54 class Puzzle {
55 public:
56     char map[MAX_X][MAX_Y];
57     struct Field field[MAX_FIELD];
58     int cnt;
59
60 public:
61     Puzzle();
62     void setField(char* str);
63     void setIntersect(Word* list);
64     void setDomain(Word* list);
65     void forwardCheck();
66     void AC_3();
67     void AC_3(bool ddomain);
68     int nodeExpand(std::stack<Node> &s, Node &n);
69     int backtrack();
70     int searchMRV(Node n);
71     void printPuzzle();
72 };
73
74 bool intersect(Field a, Field b);
75 unsigned int countTrue(bool* b, int n);
76 void printStack(std::stack<Node> s); // debug
77
78 // the compare operator of the priority queue
79 struct MRV_Cmp {
19
20 fclose(fp);
21
22 char buf[500];
23 char* ptr;
24
25 fp = fopen("./puzzle.txt", "r");
26
27 while(fgets(buf, sizeof(buf), fp)) {
28     struct Puzzle puzzle;
29
30     puzzle.setField(buf);
31     puzzle.setDomain(word_list); // check unary constraint
32     puzzle.setIntersect();
33     puzzle.AC_3();
34     if(puzzle.backtrack() != -1) {
35         puzzle.printPuzzle();
36         printf("\n-----success-----\n");
37     }
38     else {
39         puzzle.printPuzzle();
40         fprintf(stderr, "the puzzle has no solution\n");
41     }
42     // break;
43 }
44
45 fclose(fp);
46
47 return 0;
48 }
49 }
```

main.c + puzzle.h (2)

```
File Edit Selection Find View Goto Tools Project Preferences Help
puzzle.cpp
217 }
218
219 // return the fid of field which have minimum d_cnt
220 int Puzzle::searchMRV(Node n) {
221     std::vector<int> arr;
222     int mrv = INT_MAX;
223     int var;
224     for(int i=0; i<cnt; i++) {
225         if(!n.expanded[i+1] && (n.d_cnt[i] < mrv)) {
226             // printf("%d %d %d\n", i, mrv, n.d_cnt[i]);
227             mrv = n.d_cnt[i];
228             var = i;
229         }
230     }
231     return var;
232 }
233
234 void Puzzle::printPuzzle() {
235     for(int i=0; i<cnt; i++) {
236         Field* f = &field[i];
237         int _x = 0;
238         int _y = 0;
239         if(f->dir == 'A') { _x = 1; }
240         if(f->dir == 'D') { _y = 1; }
241
242         for(int j=0; j<f->length; j++) {
243             map[f->start_y + (_y*j)][f->start_x + (_x*j)] = f->val[j];
244         }
245         printf("%3d %s\n", i, f->val);
246     }
247
248     // print
249     for(int i=0; i<MAX_X+1; i++) printf(" ");
250     printf("\n");
251     for(int i=0; i<MAX_Y; i++) {
252         printf("|");
253         for(int j=0; j<MAX_X; j++) {
254             if(map[i][j] == 0) printf(" ");
255             else printf(" %c", map[i][j]);
256         }
257         printf("\n");
258     }
259 }
```

```
File Edit Selection Find View Goto Tools Project Preferences Help
puzzle.cpp
250 printf("\n");
251 for(int i=0; i<MAX_Y; i++) {
252     printf("\n");
253     for(int j=0; j<MAX_X; j++) {
254         if(map[i][j] == 0) printf(" ");
255         else printf("%c", map[i][j]);
256     }
257     printf("\n");
258 }
259 for(int i=0; i<MAX_X+1; i++) printf(" ");
260 printf("\n");
261 return;
262 }
263
264 bool intersect(Field a, Field b) {
265     if(a.dir == b.dir) return false;
266     if((a.start_x > b.start_x) && (a.start_y > b.start_y)) return false;
267     if((a.start_x < b.start_x) && (a.start_y < b.start_y)) return false;
268     return true;
269 }
270
271
272 unsigned int countTrue(bool* b, int size) {
273     unsigned int count = 0;
274     for(int i=0; i<size; i++) {
275         count += b[i] & 1;
276     }
277     return count;
278 }
279
280 void printStack(std::stack<Node> s) {
281     while(!s.empty()) {
282         printf("%s, %d\n", s.top().val, s.top().d_cnt[s.top().fid]);
283         s.pop();
284     }
285     printf("- - - - \n");
286 }
```

Line 151, Column 1 53 misspelled words Tab Size: 4 C++

```
File Edit Selection Find View Goto Tools Project Preferences Help
puzzle.cpp
112 // return value: (-1)fail, (0)had expanded, (1)succes
113 int Puzzle::nodeExpand(std::stack<Node> &s, Node &n) {
114     int fail = 0;
115     // printStack(s);
116
117     if(n.expanded[n.fid+1]) {
118         if(!n.root) { // remove value in the field(*)
119             memset(field[n.fid].val, 0, STR_SIZE);
120         }
121         return 0;
122     }
123
124     if(n.root) {
125         n.expanded[0] = true;
126         return 1;
127     }
128
129     // domain contains only the assigned value
130     memset(n.domain[n.fid], 0, sizeof(n.domain[n.fid]));
131     n.domain[n.fid][n.wid] = true;
132     n.d_cnt[n.fid] = 1;
133     // inferences(forward)
134     // consistency check: empty domain(forward)
135     // consistency check: intersection
136     for(int i=0; i<field[n.fid].length; i++) {
137         char* c_ptr = field[n.fid].itrsc[i]; // pointer to the intersected char
138         char c_val = n.val[i]; // current char value
139         // printf("-----%d-%d----\n", n.fid, i);
140         if(c_ptr == NULL) continue; // no intersection
141         else if(*c_ptr == 0 || *c_ptr == c_val) continue; // not assigned yet or same
142         // printf("(%c, %c)\n", *c_ptr, c_val);
143         return -1;
144     }
145
146     // store the value of the current node
147     // if(n.fid == 0)
148     //     printf("field %d: %s\n", n.fid, n.val);
149     strncpy(field[n.fid].val, n.val, STR_SIZE);
150     n.expanded[n.fid+1] = true;
151     return 1;
152 }
```

Line 151, Column 1 53 misspelled words Tab Size: 4 C++

File Edit Selection Find View Goto Tools Project Preferences Help

```
puzzle.cpp
154 int Puzzle::backtrack() {
155     std::stack<Node> s;
156     int depth = 0;
157
158     // push root before search
159     Node root;
160     memset(&root, 0, sizeof(root));
161     for(int i=0; i<cnt; i++) {
162         memcpy(root.domain[i], field[i].domain, sizeof(root.domain[i]));
163         root.d_cnt[i] = field[i].d_cnt;
164     }
165     root.fid = -1;
166     root.wid = -1;
167     root.root = true;
168
169     s.push(root);
170
171
172     while(!s.empty()) {
173         Node curr = s.top();
174         s.pop();
175         // node expansion
176         int flag = nodeExpand(s, curr);
177         if(flag == -1) { // fail expansion
178             continue;
179         }
180         else if(flag == 0) { // already expanded -> next child
181             depth--;
182             // printf("%d depth\n", depth);
183             continue;
184         }
185         else
186             s.push(curr);
187
188         // all variable has been assigned
189         if(depth == cnt) return 0;
190
191         // choose the next field to expand
192         // Field* next = &field[depth];
193         Field* next = &field[searchMRV(curr)];
194         // generate children for each value in the domain
```

Line 151, Column 1

53 misspelled words

Tab Size: 4

C++

File Edit Selection Find View Goto Tools Project Preferences Help

```
puzzle.cpp
181     depth--;
182     // printf("%d depth\n", depth);
183     continue;
184 }
185 else
186     s.push(curr);
187
188 // all variable has been assigned
189 if(depth == cnt) return 0;
190
191 // choose the next field to expand
192 // Field* next = &field[depth];
193 Field* next = &field[searchMRV(curr)];
194 // generate children for each value in the domain
195 for(int i=0; i<MAX_DOMAIN; i++) {
196     if(next->domain[i] == false) continue;
197     Node child;
198     child.fid = next->id;
199     child.wid = i;
200     strncpy(child.val, word_list[i].str, STR_SIZE);
201     for(int j=0; j<cnt; j++) {
202         memcpy(child.domain[j], curr.domain[j], sizeof(child.domain[j]));
203         child.d_cnt[j] = curr.d_cnt[j];
204         child.expanded[j+1] = curr.expanded[j+1];
205     }
206     child.root = false;
207     s.push(child);
208     // printf("%s\n", child.val);
209 }
210 depth++;
211 // printf("stack size: %ld\n", s.size());
212 // printfStack(s);
213 // return 0;
214 }
215
216 return -1; // stack become empty
217 }
218
219 // return the fid of field which have minimum d_cnt
220 int Puzzle::searchMRV(Node n) {
221     std::vector<int> v;
```

Line 151, Column 1

53 misspelled words

Tab Size: 4

C++