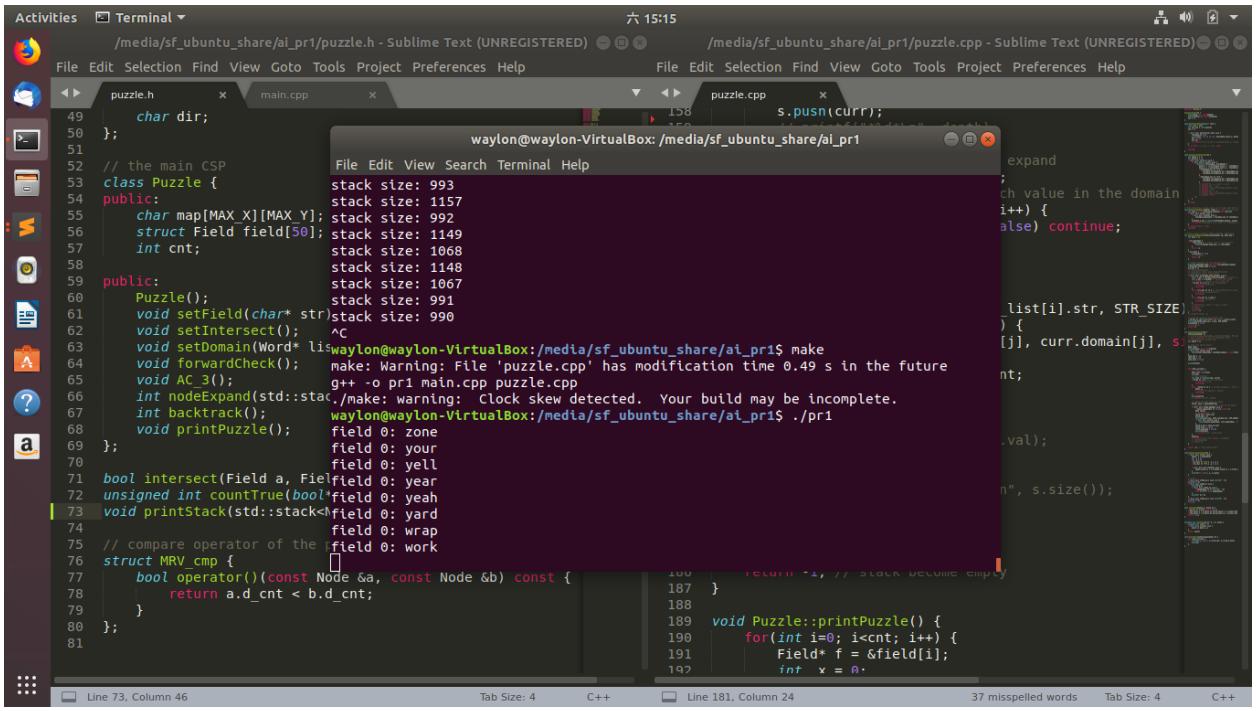
AI Project 1 Report

0516076 施威綸

Tests

- ### Brute force (node consistent)

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



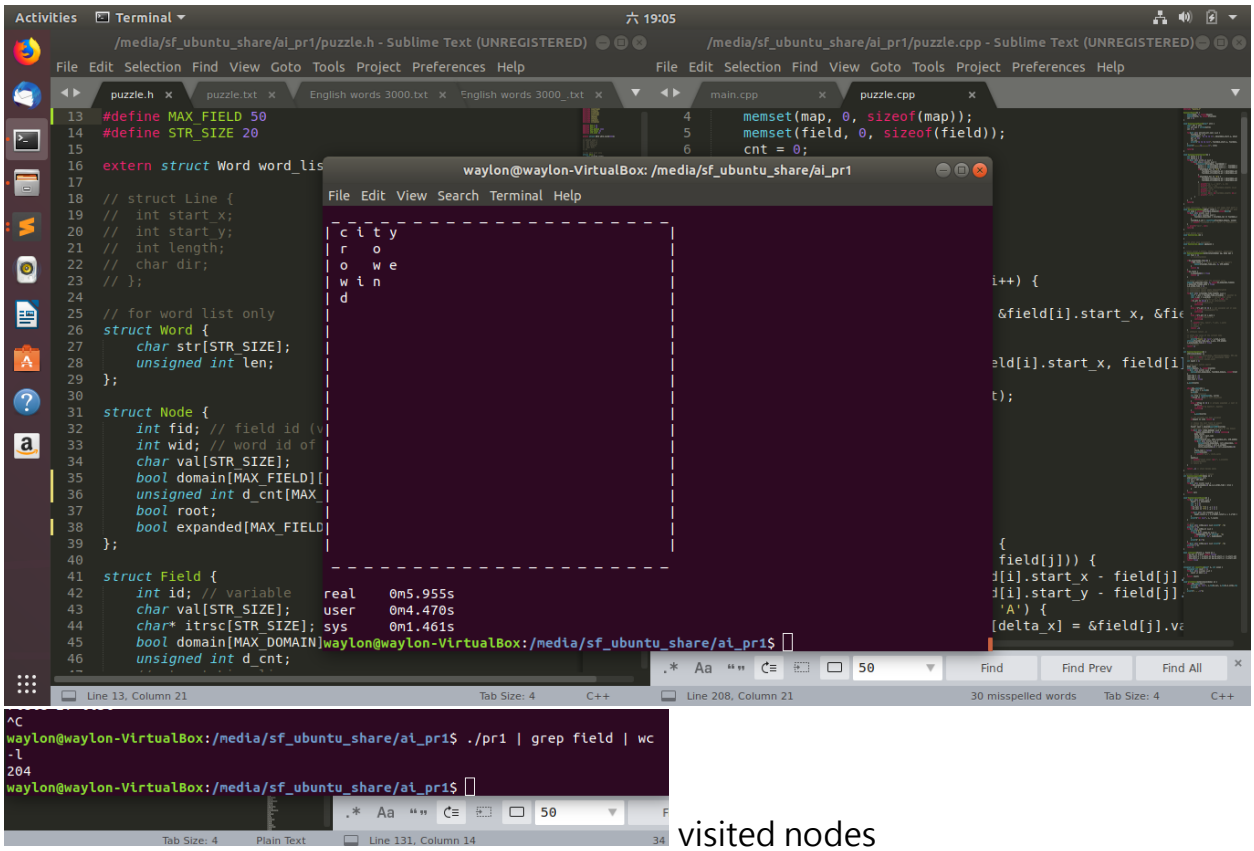
  The time complexity is $O(b^n)$, where the branching factor here is 500 approximately, the time to complete searching is all by luck.

  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.



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.

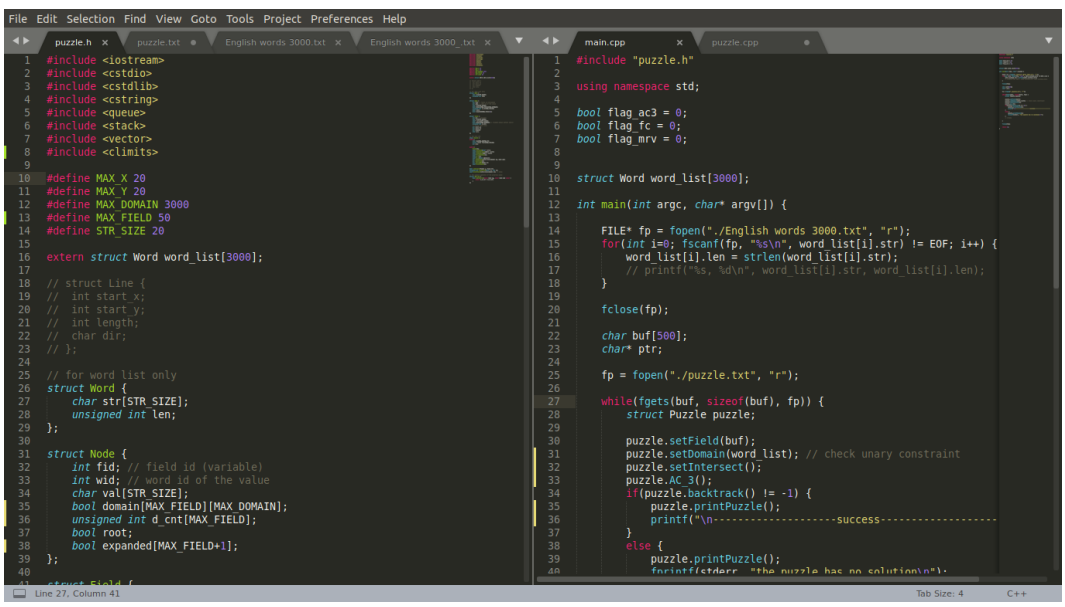
- **Observation**

  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



```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <cstring>
#include <queue>
#include <stack>
#include <vector>
#include <climits>

#define MAX_X 20
#define MAX_Y 20
#define MAX_DOMAIN 3000
#define MAX_FIELD 50
#define STR_SIZE 20

extern struct Word word_list[3000];

// 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 the value
    char val[STR_SIZE];
    bool domain[MAX_FIELD][MAX_DOMAIN];
    unsigned int d_cnt[MAX_FIELD];
    bool root;
    bool expanded[MAX_FIELD+1];
};
```

```
#include "puzzle.h"

using namespace std;

bool flag_ac3 = 0;
bool flag_fc = 0;
bool flag_mrv = 0;


struct Word word_list[3000];

int main(int argc, char* argv[]) {

    FILE* fp = fopen("./English words 3000.txt", "r");
    for(int i=0; fscanf(fp, "%s\n", word_list[i].str) != EOF; i++) {
        word_list[i].len = strlen(word_list[i].str);
        // printf("%s, %d\n", word_list[i].str, word_list[i].len);
    }

    fclose(fp);

    char buf[500];
    char* ptr;

    fp = fopen("./puzzle.txt", "r");

    while(fgets(buf, sizeof(buf), fp)) {
        struct Puzzle puzzle;

        puzzle.setField(buf);
        puzzle.setDomain(word_list); // check unary constraint
        puzzle.setIntersect();
        puzzle.AC_3();
        if(puzzle.backtrack() != -1) {
            puzzle.printPuzzle();
            printf("\n-------------------success------------------
        }
        else {
            puzzle.printPuzzle();
            fprintf(stderr, "the puzzle has no solution\n");
```

main.c + puzzle.h (1)

File Edit Selection Find View Goto Tools Project Preferences Help

puzzle.h | puzzle.txt | English words 3000.txt | English words 3000_.txt

```cpp
struct Field {
    int id; // variable
    char val[STR_SIZE];
    char* itrsc[STR_SIZE];
    bool domain[MAX_DOMAIN]; // trimmed domain before search
    unsigned int d_cnt;
    // struct Line line;
    int start_x;
    int start_y;
    int length;
    char dir;
};

// the main CSP
class Puzzle {
public:
    char map[MAX_X][MAX_Y];
    struct Field field[MAX_FIELD];
    int cnt;

public:
    Puzzle();
    void setField(char* str);
    void setIntersect(Word* list);
    void setDomain(Word* list);
    void forwardCheck();
    void AC_3();
    void AC_3(bool &domain);
    int nodeExpand(std::stack<Node> &s, Node &n);
    int backtrack();
    int searchMRV(Node n);
    void printPuzzle();
};

bool intersect(Field a, Field b);
unsigned int countTrue(bool* b, int n);
void printStack(std::stack<Node> s); // debug

// the compare operator of the priority queue
struct MRV_cmp {
```

Line 27, Column 41 | Tab Size: 4 | C++

main.cpp | puzzle.cpp

```cpp
    fclose(fp);

    char buf[500];
    char* ptr;

    fp = fopen("./puzzle.txt", "r");

    while(fgets(buf, sizeof(buf), fp)) {
        struct Puzzle puzzle;

        puzzle.setField(buf);
        puzzle.setDomain(word_list); // check unary constraint
        puzzle.setIntersect();
        puzzle.AC_3();
        if(puzzle.backtrack() != -1) {
            puzzle.printPuzzle();
            printf("\n--------------------success-------------------
        }
        else {
            puzzle.printPuzzle();
            fprintf(stderr, "the puzzle has no solution\n");
        }
        // break;

    }

    fclose(fp);

    return 0;
}
```

main.c + puzzle.h (2)

File Edit Selection Find View Goto Tools Project Preferences Help

puzzle.cpp

```cpp
}

// return the fid of field which have minimum d_cnt
int Puzzle::searchMRV(Node n) {
    std::vector<int> arr;
    int mrv = INT_MAX;
    int var;
    for(int i=0; i<cnt; i++) {
        if(!n.expanded[i+1] && (n.d_cnt[i] < mrv)) {
            // printf("%d %d %d\n", i, mrv, n.d_cnt[i]);
            mrv = n.d_cnt[i];
            var = i;
        }
    }
    return var;
}

void Puzzle::printPuzzle() {
    for(int i=0; i<cnt; i++) {
        Field* f = &field[i];
        int _x = 0;
        int _y = 0;
        if(f->dir == 'A') { _x = 1; }
        if(f->dir == 'D') { _y = 1; }

        for(int j=0; j<f->length; j++) {
            map[f->start_y + (_y*j)][f->start_x + (_x*j)] = f->val[j];
        }
        printf("%3d %s\n", i, f->val);
    }

    // print
    for(int i=0; i<MAX_X+1; i++) printf(" _");
    printf("\n");
    for(int i=0; i<MAX_Y; i++) {
        printf("|");
        for(int j=0; j<MAX_X; j++) {
            if(map[i][j] == 0) printf("  ");
            else printf(" %c", map[i][j]);
        }
        printf(" |\n");
```

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

File Edit Selection Find View Goto Tools Project Preferences Help

puzzle.cpp

```cpp
    printf("\n");
    for(int i=0; i<MAX_Y; i++) {
        printf("|");
        for(int j=0; j<MAX_X; j++) {
            if(map[i][j] == 0) printf("  ");
            else printf(" %c", map[i][j]);
        }
        printf(" |\n");
    }
    for(int i=0; i<MAX_X+1; i++) printf(" _");
    printf("\n");
    return;
}

bool intersect(Field a, Field b) {
    if(a.dir == b.dir) return false;
    if((a.start_x > b.start_x) && (a.start_y > b.start_y)) return false;
    if((a.start_x < b.start_x) && (a.start_y < b.start_y)) return false;
    return true;
}


unsigned int countTrue(bool* b, int size) {
    unsigned int count = 0;
    for(int i=0; i<size; i++) {
        count += b[i] & 1;
    }
    return count;
}

void printStack(std::stack<Node> s) {
    while(!s.empty()) {
        printf("%s, %d\n", s.top().val, s.top().d_cnt[s.top().fid]);
        s.pop();
    }
    printf("- - - - -\n");
}
```

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

File Edit Selection Find View Goto Tools Project Preferences Help

puzzle.cpp

```cpp

// return value: (-1)fail, (0)had expanded, (1)success
int Puzzle::nodeExpand(std::stack<Node> &s, Node &n) {
    int fail = 0;
        // printStack(s);

    if(n.expanded[n.fid+1]) {
        if(!n.root) { // remove value in the field(*)
            memset(field[n.fid].val, 0, STR_SIZE);
        }
        return 0;
    }
    if(n.root) {
        n.expanded[0] = true;
        return 1;
    }

    // domain contains only the assigned value
    memset(n.domain[n.fid], 0, sizeof(n.domain[n.fid]));
    n.domain[n.fid][n.wid] = true;
    n.d_cnt[n.fid] = 1;
    // inferences(forward)
    // consistency check: empty domain(forward)
    // consistency check: intersection
    for(int i=0; i<field[n.fid].length; i++) {
        char* c_ptr = field[n.fid].itrsc[i]; // pointer to the intersected char
        char c_val = n.val[i]; // current char value
        // printf("-----%d-%d-----\n", n.fid, i);
        if(c_ptr == NULL) continue; // no intersection
        else if(*c_ptr == 0 || *c_ptr == c_val) continue; // not assigned yet or same
        // printf("(%c, %c)\n", *c_ptr, c_val);
        return -1;
    }

    // store the value of the current node
    // if(n.fid == 0)
        // printf("field %d: %s\n", n.fid,n.val);
    strncpy(field[n.fid].val, n.val, STR_SIZE);
    n.expanded[n.fid+1] = true;
    return 1;
}
```

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

File Edit Selection Find View Goto Tools Project Preferences Help

puzzle.cpp

```cpp
int Puzzle::backtrack() {
    std::stack<Node> s;
    int depth = 0;

    // push root before search
    Node root;
    memset(&root, 0, sizeof(root));
    for(int i=0; i<cnt; i++) {
        memcpy(root.domain[i], field[i].domain, sizeof(root.domain[i]));
        root.d_cnt[i] = field[i].d_cnt;
    }
    root.fid = -1;
    root.wid = -1;
    root.root = true;

    s.push(root);


    while(!s.empty()) {
        Node curr = s.top();
        s.pop();
        // node expansion
        int flag = nodeExpand(s, curr);
        if(flag == -1) { // fail expansion
            continue;
        }
        else if(flag == 0) { // already expanded -> next child
            depth--;
            // printf("%d depth\n", depth);
            continue;
        }
        else
            s.push(curr);

        // all variable has been assigned
        if(depth == cnt) return 0;

        // choose the next field to expand
        // Field* next = &field[depth];
        Field* next = &field[searchMRV(curr)];
```

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

File Edit Selection Find View Goto Tools Project Preferences Help

puzzle.cpp

```cpp
            depth--;
            // printf("%d depth\n", depth);
            continue;
        }
        else
            s.push(curr);

        // all variable has been assigned
        if(depth == cnt) return 0;

        // choose the next field to expand
        // Field* next = &field[depth];
        Field* next = &field[searchMRV(curr)];
        // generate children for each value in the domain
        for(int i=0; i<MAX_DOMAIN; i++) {
            if(next->domain[i] == false) continue;
            Node child;
            child.fid = next->id;
            child.wid = i;
            strncpy(child.val, word_list[i].str, STR_SIZE);
            for(int j=0; j<cnt; j++) {
                memcpy(child.domain[j], curr.domain[j], sizeof(child.domain[j]));
                child.d_cnt[j] = curr.d_cnt[j];
                child.expanded[j+1] = curr.expanded[j+1];
            }
            child.root = false;
            s.push(child);
            // printf("%s\n", child.val);
        }
        depth++;
        // printf("stack size: %ld\n", s.size());
        // printStack(s);
        // return 0;
    }

    return -1; // stack become empty
}

// return the fid of field which have minimum d_cnt
int Puzzle::searchMRV(Node n) {
```

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