Τεχνητή Νοημοσύνη :
1η Προαιρετική Εργασία 2022

Υφαντίδης Δημήτρης, ΑΕΜ: 3938

1. Εισαγωγή

Η Συγκεκριμένη εργασία υλοποιήθηκε στη γλώσσα C++, στο πρότυπο C++20. Όλα τα απαραίτητα αρχεία κώδικα για την μεταγλώττιση και την παραγωγή του εκτελέσιμου βρίσκονται στον φάκελο «Κώδικας».

Για την υλοποίηση χρησιμοποιήθηκαν διάφορες σύγχρονες δομές της STL για ευκολία συγγραφής και κατανόησης κώδικα, καθώς και παλαιότερα χαρακτηριστικά της C για οικονομία χώρου και χρόνου. Οι μετρικές της απόδοσης και της εκτέλεσης του αλγορίθμου, καθώς και η οπτική αναπαράσταση της λύσης του παζλ καταγράφονται, άνευ απρόοπτου, σε ένα αρχείο «results.txt».

Στην παρακάτω τεχνική έκθεση θα αναφερθεί συνοπτικά και σταδιακά η μοντελοποίηση του δεδομένου προβλήματος. Αρχικά, θα αναφερθεί η μοντελοποίηση των χρωματιστών υγρών, μετά των δοχείων, μετά των καταστάσεων και μετά του αλγορίθμου BFS και της main. Σε κάθε παράγραφο θα αναφέρονται και τα σχετικά αρχεία.

Να σημειωθεί ότι σε περίπτωση που μπορεί να υπάρξει κάποια ασάφεια, υπάρχουν αρκετά και πιο αναλυτικά σχόλια στα αρχεία του κώδικα που παρέχουν μικρές λεπτομέρειες της υλοποίησης.

2. Τα χρωματιστά υγρά

Ο μέγιστος αριθμός των μπουκαλιών είναι 19 (εκφώνηση: $N < 20$), άρα ο μέγιστος αριθμός διαφορετικών χρωμάτων είναι 17 ($N_{\chi\rho\omega\mu\acute{\alpha}\tau\omega\nu} = N_{\delta o\chi\epsilon\acute{\iota}\omega\nu} - 2$). Κάθε χρώμα αντιστοιχεί σε έναν ακέραιο του τύπου δεδομένων char, από το 0 (μαύρο) έως και το 16 (κίτρινο). Η απουσία σε ένα στρώμα του δοχείου αναπαρίσταται από την τιμή -1 (μακροεντολή: NO_COLOR). Όλα τα χρώματα είναι ορισμένα (και η απουσία χρώματος) είναι ορισμένα ως μακροεντολές στο αρχείο κεφαλίδας «colors.h», μαζί με τον

πίνακα συμβολοσειρών COLOR_STR, όπου το COLOR_STR[**i**] επιστρέφει το χρώμα με κωδικό **i** σαν συμβολοσειρά.

3. Τα δοχεία

Κάθε δοχείο αναπαρίσταται από ένα αντικείμενο της κλάσης **Bottle**, που είναι δηλωμένη και υλοποιημένη στα αρχεία «Bottle.h» και «Bottle.cpp» αντίστοιχα. Κάθε δοχείο αποτελείται από έναν στατικό πίνακα 4 χαρακτήρων (**contents[4]**). Εκεί αποθηκεύονται τα χρωματιστά υγρά. Η κλάση περιέχει μεθόδους όπως:

- **bool hasFreeSpace():** Επιστρέφει αν είναι γεμάτο ή όχι
- **bool isEmpty():** Επιστρέφει αν είναι άδειο ή όχι
- **bool isComplete():** Επιστρέφει true αν είναι γεμάτο έως την κορυφή με το ίδιο υγρό ή αν είναι άδειο.
- **bool shouldPourTo(const Bottle &):** Επιστρέφει αν γίνεται το μπουκάλι να χύσει υγρό στο άλλο ή όχι. Το μπουκάλι μπορεί να χύσει k mL συνεχόμενου υγρού, αν το άλλο είναι άδειο ή έχει στην κορυφή του το ίδιο χρώμα και έχει k mL ελεύθερο χώρο.
- **color_t top():** Επιστρέφει το χρώμα του υγρού στην πρώτη στρώση, η αντίστοιχη υπερφόρτωση «top(int &)» επιστρέφει με αναφορά τον ελεύθερο χώρο. *(color_t = char με τη χρήση της εντολής typedef)*
- **color_t pour(Bottle &):** Αν γίνεται, χύνει το υγρό της κορυφής του στο αναφερόμενο μπουκάλι και επιστρέφει το χρώμα που ανταλλάχθηκε, αλλιώς επιστρέφει NO_COLOR και δεν γίνεται τίποτα.
- **color_t getColor(size_t i):** Επιστρέφει το contents[i].

+ Τελεστές (==, =, [])

Η παραπάνω κλάση και οι συναρτήσεις τής συνεισφέρουν στην ευκολία ανάγνωσης και συγγραφής του κώδικα για την μοντελοποίηση των καταστάσεων του παζλ.

4. Οι καταστάσεις

Κάθε στιγμιότυπο του παζλ αναπαρίσταται από ένα αντικείμενο της κλάσης **State** των αρχείων «State.h» και «State.cpp». Κάθε κατάσταση αποτελείται από έναν δείκτη σε πίνακα δοχείων, το πλήθος των μπουκαλιών, την λεκτική περιγραφή της μετάβασης από την προηγούμενη κατάσταση στη νυν και έναν δείκτη στην προηγούμενη κατάσταση. Σημαντικές μέθοδοι της κλάσης State αποτελούν οι:

- **void init():** Αρχικοποιεί την τα N – 2 πρώτα μπουκάλια της αρχικής κατάστασης με N – 2 τυχαία υγρά (4 mL το καθένα) με τυχαία σειρά. Οι τυχαίες επιλογές γίνονται με τη χρήση της γεννήτριας Mersenne Twister (std::mt19937 της βιβλιοθήκης <random>).
- **hash_t hashValue():** Επιστρέφει την τιμή κατακερματισμού του αντικειμένου. Ο μηχανισμός υπολογισμού της έγινε σύμφωνα με το πρότυπο της υπερφόρτωσης της συνάρτησης hashCode() των αντικειμένων της γλώσσας Java. *(hash_t = long long με typedef).*
- **std::string toString():** Επιστρέφει μια οπτική αναπαράσταση μιας κατάστασης του προβλήματος π.χ:

```
   [1]        [2]        [3]        [4]        [5]        [6]        [7]        [8]
|PINK   |  |INDIGO |  |PINK   |  |CYAN   |  |LIME   |  |LIME   |  |       |  |       |
|CRIMSON|  |PINK   |  |BROWN  |  |BROWN  |  |BROWN  |  |PINK   |  |       |  |       |
|BROWN  |  |INDIGO |  |CRIMSON|  |CYAN   |  |LIME   |  |CYAN   |  |       |  |       |
|LIME   |  |INDIGO |  |INDIGO |  |CRIMSON|  |CRIMSON|  |CYAN   |  |       |  |       |
---------  ---------  ---------  ---------  ---------  ---------  ---------  ---------
```

Εικόνα 1

```
   [1]        [2]        [3]        [4]        [5]        [6]        [7]        [8]
|       |  |INDIGO |  |       |  |CYAN   |  |LIME   |  |LIME   |  |       |  |       |
|       |  |PINK   |  |CRIMSON|  |BROWN  |  |BROWN  |  |PINK   |  |       |  |       |
|BROWN  |  |INDIGO |  |CRIMSON|  |CYAN   |  |LIME   |  |CYAN   |  |PINK   |  |       |
|LIME   |  |INDIGO |  |INDIGO |  |CRIMSON|  |CRIMSON|  |CYAN   |  |PINK   |  |BROWN  |
---------  ---------  ---------  ---------  ---------  ---------  ---------  ---------
```

Εικόνα 2

- **bool isVictorious():** Επιστρέφει true αν η κατάσταση είναι τέτοια έτσι ώστε το παζλ να ολοκληρώθηκε, αλλιώς false. Η συνάρτηση αυτή αποτελεί την συνθήκη τερματισμού του BFS.
- **int getDepth():** Επιστρέφει το βάθος του στιγμιότυπου στο δένδρο των καταστάσεων.

- **color_t pour(State &n, int i, int j):** Ελέγχει αν γίνεται να χυθεί υγρό από το i-οστό δοχείο στο j-οστό δοχείο. Αν ναι, δημιουργείται και αποθηκεύεται στο **n** μια καινούργια κατάσταση ίδια με την τωρινή όπου το δοχείο **i** έχει χύσει στο δοχείο **j**, εισάγεται στο δένδρο καταστάσεων και επιστρέφεται το χρώμα του υγρού που χύθηκε. Αλλιώς δεν γίνεται τίποτα και επιστρέφει NO_COLOR.
  Η εν λόγω συνάρτηση αποτελεί τον τελεστή μετάβασης μεταξύ των καταστάσεων.

- **std::vector<State *> *expand():** Επιστρέφει τις καταστάσεις παιδιά, δηλαδή κάθε κατάσταση που μπορεί να εφαρμοστεί επιτυχώς ο τελεστής μετάβασης από κάθε μπουκάλι σε κάθε άλλο μπουκάλι.

5. Ο BFS και το κυρίως πρόγραμμα

Το κυρίως πρόγραμμα (**int main() { … }**) βρίσκεται μαζί με τη συνάρτηση υλοποίησης του αλγορίθμου αναζήτησης πρώτα σε πλάτος (BFS) στο αρχείο «main.cpp».

Για την υλοποίηση του BFS χρησιμοποιήθηκαν οι δομές:

std::queue<State *> frontier για το μέτωπο αναζήτησης (ουρά)

std::unordered_map<hash_t, State *> closed για το κλειστό σύνολο ως πίνακα κατακερματισμού.

Στον BFS μεταβιβάζεται η αρχική κατάσταση και εισάγεται στο μέτωπο αναζήτησης. Επίσης μεταβιβάζονται με αναφορά οι μεταβλητές **examined** και **memory** όπου αποθηκεύονται ο αριθμός των καταστάσεων που εξετάστηκαν και ο αριθμός των συνολικών καταστάσεων αντίστοιχα. Η υλοποίηση φαίνεται στην επόμενη σελίδα.

```
State *BFS(State& initial, unsigned long long &examined, unsigned long long &memory)
{
    std::queue<State *> frontier;
    std::unordered_map<hash_t, State *> closed;
    std::vector<State *> *children;

    frontier.push(&initial);
    examined = 0;
    memory = 1;

    while (!frontier.empty())
    {
        if (frontier.size() + closed.size() > memory) {
            memory = frontier.size() + closed.size();
        }
        State *s = frontier.front();

        frontier.pop();

        if (closed.count(s->hashValue()) == 0)
        {
            examined += 1;

            if (s->isVictorious()) {
                // Goal state reached.
                return s;
            }
            closed.insert({s->hashValue(), s});

            children = s->expand();

            for (const auto& child : *children)
            {
                if (closed.count(child->hashValue()) == 0) {
                    frontier.push(child);
                }
            }
            delete children;
        }
    }
    return nullptr;
}
```

Όσον αφορά το κυρίως πρόγραμμα:

- Η μακροεντολή **BOTTLES_N** καθορίζει τον αριθμό των μπουκαλιών. Στο παραδοτέο είναι ορισμένο με τιμή 8, για άλλο αριθμό μπουκαλιών πρέπει να επαναπροσδιοριστεί και να μεταγλωττιστεί ξανά το πρόγραμμα.
- Δηλώνονται οι μεταβλητές **memory** και **examined**.
- Δημιουργείται και ανοίγει ένα αρχείο «results.txt» όπου θα εκτυπωθούν οι μετρικές και η οπτική αναπαράσταση των καταστάσεων και των μεταβάσεων της λύσης. (Σε περίπτωση που κάτι πάει λάθος με το αρχείο, τα αποτελέσματα εκτυπώνονται στην κονσόλα).
- Δηλώνονται και κάποια επιπλέον αντικείμενα για την καταγραφή των χρονοσφραγίδων αρχής και λήξης του αλγορίθμου, μαζί με τη μεταβλητή **duration** που αποθηκεύει τον χρόνο εκτέλεσης του αλγορίθμου BFS σε milliseconds. Αυτό αποτελεί μια έξτρα μετρική απόδοσης του αλγορίθμου.

6. Αποτελέσματα

Τα αποτελέσματα διαφέρουν από εκτέλεση σε εκτέλεση για σταθερό αριθμό δοχείων. Παρακάτω καταγράφονται τα αποτελέσματα 3 διαφορετικών εκτελέσεων για N = 6, N = 8, N = 10 και N = 11.

<u>Για 6 δοχεία:</u>

Μέσος Χώρος: αμελητέος

Μέσος Χρόνος: ½ δευτερόλεπτο

```
METRICS FOR 6 BOTTLES:
-> Depth:           9
-> Total Nodes:     5233
-> Examined Nodes:  2353
-> Elapsed Time:    00h : 00m : 00s : 025ms
```

```
METRICS FOR 6 BOTTLES:
-> Depth:           13
-> Total Nodes:     11621
-> Examined Nodes:  5661
-> Elapsed Time:    00h : 00m : 00s : 062ms
```

```
METRICS FOR 6 BOTTLES:
-> Depth:           9
-> Total Nodes:     5233
-> Examined Nodes:  2353
-> Elapsed Time:    00h : 00m : 00s : 025ms
```

Για 8 δοχεία:

Μέσος Χώρος (με βάση τη διαχείριση εργασιών): ~100 έως 300 MB

Μέσος Χρόνος: ~1 με 2 δευτερόλεπτα

```
METRICS FOR 8 BOTTLES:
-> Depth:           15
-> Total Nodes:     200878
-> Examined Nodes:  90601
-> Elapsed Time:    00h : 00m : 01s : 573ms
```

```
METRICS FOR 8 BOTTLES:
-> Depth:           16
-> Total Nodes:     237982
-> Examined Nodes:  93267
-> Elapsed Time:    00h : 00m : 01s : 630ms
```

```
METRICS FOR 8 BOTTLES:
-> Depth:           17
-> Total Nodes:     308356
-> Examined Nodes:  127079
-> Elapsed Time:    00h : 00m : 02s : 182ms
```

Για 10 δοχεία:

Μέσος Χώρος (με βάση τη διαχείριση εργασιών): ~3.5 έως 5 GB μνήμης

Μέσος Χρόνος: ~1 έως 1.5 λεπτό

```
METRICS FOR 10 BOTTLES:
-> Depth:           20
-> Total Nodes:     2184986
-> Examined Nodes:  687382
-> Elapsed Time:    00h : 00m : 18s : 191ms
```

```
METRICS FOR 10 BOTTLES:
-> Depth:           23
-> Total Nodes:     9350294
-> Examined Nodes:  3202919
-> Elapsed Time:    00h : 01m : 23s : 650ms
```

```
METRICS FOR 10 BOTTLES:
-> Depth:           25
-> Total Nodes:     12268311
-> Examined Nodes:  5015040
-> Elapsed Time:    00h : 02m : 08s : 291ms
```

Για 11 δοχεία:

Μέσος Χώρος (με βάση τη διαχείριση εργασιών και την αναφορά χωρητικότητας του SSD): ~28 GB μνήμης (12 GB peak RAM + 16 στον SSD)

Μέσος Χρόνος: ~12 με 15 λεπτά

```
METRICS FOR 11 BOTTLES:
-> Depth:           25
-> Memory:          67066607
-> Examined:        22334518
-> Elapsed Time:    00h : 12m : 16s : 100ms
```

Για περισσότερα δοχεία:

Η διαφορά χρόνου μεγαλώνει πολύ από τα 8 μπουκάλια στα 10 και από τα 10 στα 11. Δεδομένου αυτών των μετρικών, θα ήταν πολύ δύσκολο να βρεθεί λύση για N = 12 και πρακτικά αδύνατον από άποψη χρόνου και μνήμης για N > 14. Ωστόσο ο αλγόριθμος BFS είναι τυφλός επομένως άλλοι αλγόριθμοι τεχνητής νοημοσύνης να αποδίδουν καλύτερα. Επιπλέον για περισσότερα δοχεία, ενδεχομένως να υπάρχουν περισσότερες συγκρούσεις με τις τιμές κατακερματισμού των καταστάσεων.