TCG - HW1 Report

1. Implementation

- **How to compile and run [bi-direction BFS (bit)]**
  - compile : 在makefile那層的資料夾中輸入 : make bds
  - run : compile完成後在makefile那層的資料夾中輸入 :
    - time ./shiroBDS < [input file] > [output file] (with execution time)
    - ./shiroBDS < [input file] > [output file] (without execution time)
- **Algorithm Implementation**
  - BFS (naive)
  - BFS (bit)
  - bi-direction BFS (naive)
  - bi-direction BFS (bit)
- **Queue Implementation (bit)**
  - Node的內容存一個 unsigned long long int (50bit地圖 6bit玩家位置 4bit這一步的移動策略 1bit是在奇數步or偶數步(for bi-direction bfs決定何時交換搜索方向))

```
class bfs_node{
    public:
        unsigned long long int playerMoveBox;
        // xxx=&&&&******(50$)
        //  = : parity bit  * : player  ,  & : move , $ : box
        bfs_node *next;

        //constructor
        bfs_node(): playerMoveBox(9), next(0){};
        bfs_node(unsigned long long int pmb): next(0){
            playerMoveBox = pmb;
        };
};
```

  - 自製的Queue中設有pool的功能，節省反覆刪除node以及建立node造成的資源浪費

```
class bfs_queue{
    public :
        bfs_node *head;
        bfs_node *tail;
        bfs_node *pool;
        bfs_node *tmp;
        int poolSize; // for analysis
        int size; // for analysis
        unsigned long long int data;
```

```
void push(unsigned long long int*);
bool isEmpty();
unsigned long long int pop();
```

- **HashMap Implementation (Naive)**
    - 使用ELFHash實作

```
int shiro_hashmap::getELFHash(char* str){
    unsigned int hash = 0;
    unsigned int x = 0;
    while(*str) {
        hash = (hash << 4) + *str;
        if ((x=hash & 0xf0000000) != 0) {
            hash ^= (x >> 24);
            hash &= ~x;
        }
        str ++;
    }
    hash &= 0x7fffffff;
    return hash;
}
```

- **HashMap Implementation (bit)**
    - 使用FNV-1 hash實作

```
bool shiro_hashmap::insert(unsigned long long int* d){
    hash = offset;
    /*for(int i=0 ; i<h_num ; ++i){
        hash *= prime;
        hash ^= (((*d) >> (8*i)) & 255);
        //hash *= prime;
    }*/
    hash *= prime;
    hash ^= ((*d) & 255);
    hash *= prime;
    hash ^= (((*d) >> (8)) & 255);
    hash *= prime;
    hash ^= (((*d) >> (16)) & 255);
    hash *= prime;
    hash ^= (((*d) >> (24)) & 255);
    hash *= prime;
    hash ^= (((*d) >> (32)) & 255);
    hash *= prime;
    hash ^= (((*d) >> (40)) & 255);
```

```
    hash *= prime;
    hash ^= (((*d) >> (48)) & 255);
    key = list[hash%bucket];
    tmp = key;
    // xxxx&&&&******(50$)
    //  x : useless , * : player  ,  & : move , $ : box
    if(!key){
        size++;
    }

    while(key){
        // board exists
        if (!(((*d)<<8)^(((key->data))<<8))) {
            return false;
        }
        key = key->next;
    }
    node = new shiro_hashmap_Node();
    node->data = (*d);
    node->next = tmp;
    list[hash%bucket] = node;
    node_num++;
    return true;
}
```

- bucket數量為了確保bucket使用率不會過高 (70%以下)
  - 正向搜尋設定為2^21個
  - 反向搜尋設定為2^24個

2. Experiment

測試的部分使用tiny.in 以及small前兩題張圖進行評測

- **BFS (Naive)**
  - tiny.in
    - execution time
      - real : 7m39.652s ｜ user : 6m26.596s ｜ sys : 0m5.339s
    - maximum memory usage(使用queue以及hashmap中的node數量進行判斷)
      - Queue (include pool) : 22427
      - HashMap nodes : 466499
  - small puzzle 1
    - execution time

        - 大於10分鐘
    - maximum memory usage(使用queue以及hashmap中的node數量進行判斷)
        - Queue (include pool) : x
        - HashMap : x
- small puzzle 2
    - execution time
        - real : 23.004s | user : 21.050s | sys : 0.269s
    - maximum memory usage(使用queue以及hashmap中的node數量進行判斷)
        - Queue (include pool) : 101576
        - HashMap : 317200

- **BFS (bit)**
    - tiny.in
        - execution time
            - real : 0.574s | user : 0.555s | sys : 0.014s
        - maximum memory usage(使用queue以及hashmap中的node數量進行判斷)
            - Queue (include pool) : 22443
            - HashMap nodes : 461286
    - small puzzle 1
        - execution time
            - real : 1.532s | user : 1.489s | sys : 0.029s
        - maximum memory usage(使用queue以及hashmap中的node數量進行判斷)
            - Queue (include pool) : 80919
            - HashMap : 1376684
    - small puzzle 2
        - execution time
            - real : 0.242s | user : 0.224s | sys : 0.011s
        - maximum memory usage(使用queue以及hashmap中的node數量進行判斷)
            - Queue (include pool) : 87944
            - HashMap : 255596
- **bi-direction BFS (Naive)**
    - tiny.in
        - execution time
            - real : 6.743s | user : 6.153s | sys : 0.097s
        - maximum memory usage(使用queue以及hashmap中的node數量進行判斷)
            - Queue (include pool) : 13793
            - HashMap nodes : 81152
    - small puzzle 1
        - execution time
            - real : 7m43.021s | user : 6m25.339s | sys : 0m5.798s

        - maximum memory usage(使用stack以及hashmap中的node數量進行判斷)
            - Stack (include pool) : 85023
            - HashMap : 567219
    - small puzzle 2
        - execution time
            - real : 0.032s ｜ user : 0.025s ｜ sys : 0.003s
        - maximum memory usage
            - Stack (include pool) : 5827
            - HashMap : 11229

- **bi-direction BFS (bit)**
    - tiny.in
        - execution time
            - real : 0.780s ｜ user : 0.576s ｜ sys : 0.194s
        - maximum memory usage(使用queue以及hashmap中的node數量進行判斷)
            - Queue (include pool) : 14072
            - HashMap : 80004
    - small puzzle 1
        - execution time
            - real : 0.667s ｜ user : 0.581s ｜ sys : 0.081s
        - maximum memory usage
            - Queue (include pool) : 82746
            - HashMap : 507446
    - small puzzle 2
        - execution time
            - real : 0.148s ｜ user : 0.069s ｜ sys : 0.076s
        - maximum memory usage
            - Queue (include pool) : 4576
            - HashMap : 8643

Result

- execution time : (依長到短 / >> 表示超過10分鐘)
    - tiny : BFS(naive) > bi-direction BFS (naive) > bi-direction BFS (bit) > BFS (bit)
    - small1 : BFS(naive) >> bi-direction BFS (naive) > BFS (bit) > bi-direction BFS (bit)
    - small2 : BFS(naive) > BFS (bit) > bi-direction BFS (bit) > bi-direction BFS (naive)

- maximum memory usage (bit):
  - tiny : BFS (bit) > bi-direction BFS (bit)
  - small1 : BFS (bit) > bi-direction BFS (bit)
  - small2 : BFS (bit) > bi-direction BFS (bit)
- maximum memory usage (naive): (超時的不列入)
  - tiny : BFS (naive) > bi-direction BFS (naive)
  - small1 : bi-direction BFS (naive)
  - small2 : BFS (naive) > bi-direction BFS (naive)

3. Discussion

- **討論experiment結果**
  - **Execution time**
    - 在地圖複雜度極低，行走限制大時(ex. small2) 由於naive方式的bi-direction BFS在儲存資料於node時有直接將走路歷史紀錄存在node中，在發現solution時不需要再回推結果，因此執行時間會較使用較小容量但歷史步數需要耗費運算轉換的方式(bit)來得快。然而，在地圖複雜時ex. small1此方式會造成存取資料的廢時，難以更快的速度進行暴力解。
    - 有關BFS以及bi-direction BFS的比較，在地圖簡單時(tiny)，使用bi-direction不一定會有比較快的效果，原因推估是在bi-direction BFS時反向搜尋的數量取決於可能的結果盤面數量，因此在地圖簡單，但可能的結果盤面數量多時會造成運算的耗時 (在層數不大時bi-direction BFS正反向各走一層，BFS可能已經走了5-10層)
  - maximum memory usage (overall)
    - naive的儲存方式是每個node中湊除存了整個地圖的狀態，加上歷史步數，因此使用的記憶體量明顯大於bit的儲存方式
    - 在同樣儲存方式中BFS和bi-direction BFS的比較 明顯BFS的用量大於 bi-direction BFS，原因是因為同樣解答為n層的題目中，BFS需要走滿n層的路線，bi-direction BFS只需要走n/2層的路線 * 出發地圖個數，在地圖層數加深路線可能性會不段加寬的狀況下，走一半層數的bi-direction BFS可以非常有效的節省記憶體使用量
- **The complexity of each algorithm**
  - 假設在某個盤面下走一步可以到達的新盤面數量為b，搜尋的深度為d
  - BFS
    - 時間複雜度
      - 假設每個Queue中的node要找到完其下一層的盤面需要時間O(e)
      - 則時間複雜度為每一層的node往下一層的盤面所需時間的總和

        $1 * e + b * e + b^2 * e + b^3 * e + \ldots + b^{d-1} * e = O(b^{d-1} * e)$
      - 在pukoban中b最大可能為12因此時間複雜度為$O(12^{d-1} * e)$
    - 空間複雜度
      - 每一走一步最多可能產生b個新盤面，每個新盤面再多走一步最多可能再產生b個新盤面，搜尋越深最多的可能盤面越多，因此盤面數量(空間複雜度)=$O(b^d)$

        - 在pukoban中b最大可能為12因此空間複雜度為$O(12^d)$
    - bi-direction BFS
        - 時間複雜度
            - 假設每個Queue中的node要找到完其下一層的盤面需要時間O(e)
            - 則時間複雜度為每一層的node往下一層的盤面所需時間的總和

              $1*e+b*e+b^2*e+b^3*e+...+b^{\frac{d}{2}-1}*e=O(b^{\frac{d}{2}-1}*e)$
            - 在pukoban中b最大可能為12因此時間複雜度為$O(12^{\frac{d}{2}-1}*e)$
        - 空間複雜度
            - 每一走一步最多可能產生b個新盤面，每個新盤面再多走一步最多可能再產生b個新盤面，搜尋越深最多的可能盤面越多，搜尋最大深度為d/2，可能的終點數量為常數可不計，因此盤面數量(空間複雜度)=$O(b^{\frac{d}{2}})$
            - 在pukoban中b最大可能為12因此空間複雜度為$O(12^d)$
- **The complexity of a Pukoban puzzle**
    - 地圖的複雜度在此以可能的盤面數量為量化標準，假設有n個非牆壁格，m個goal，m個box，1個player，則最大可能的盤面數量為：

      $C_m^n*(n-m)$

      [可能的box的位置 x 可能的玩家位置 (可能的玩家位置 = 非牆壁格數量 - box的位置)]
    - tiny.in

      Map1 : 2860 Map2 : 15015 Map3 : 30030 Map4 : 30030 Map5 : 48048 Map6 : 80080 Map7 : 80080 Map8 : 102960 Map9 : 101745 Map10 : 712530 avg : 120337 (1.2e+5)
    - small.in

      Map1 : 2260440 Map2 : 1716099 Map3 : 4604600 Map4 : 84146400 Map5 : 46823400 Map6 : 18386775 Map7 : 63109800 Map8 : 443521650 Map9 : 925610400 Map10 : 2128903920 avg : 371908348 (3.72e+8)
    - medium.in

      Map1 : 6216210 Map2 : 17760600 Map3 : 17760600 Map4 : 252439200 Map5 : 90135045 Map6 : 308864160 Map7 : 3483301360 Map8 : 2128903920 Map9 : 1693446300 Map10 : 6580248480 avg : 1457907587 (1.46e+9)
- **The complexity of your created puzzle**
    - 使用同於上列的評估方式可得複雜度為：3108105 * (28-8) = 62162100 (6.22e+7)