자료구조
숙제2 과제

2018187116
컴퓨터공학부
김용현

〈1. 큐 - 미팅 주선 프로그램〉

```cpp
/* 큐 - 미팅 주선 프로그램 */
#include <iostream>
#include <deque>
#include <string>
using namespace std;

int main(int argc, char* argv[]) {
    /* Faster */
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    /* Input */
    deque<string> m; //남성
    deque<string> f; //여성
    int n; cin >> n;

    /* Calculation */
    for (int i = 0; i < n; i++) {
        /* Init */
        int time;    //시간
        string name; //이름
        char sex;    //성별
        char fr;     //앞뒤

        /* Input */
        cin >> time >> name >> sex >> fr;

        /* Deque Insertion */
        if (sex == 'm') {
            if (fr == 'F')
                m.push_front(name);
            else if (fr == 'R')
                m.push_back(name);
             }
        else if (sex == 'f') {
            if (fr == 'F')
                f.push_front(name);
            else if (fr == 'R')
                f.push_back(name);
             }

        /* Matching */
        if (!m.empty() && !f.empty()) {
            cout << "Matched : " << m.front() << ' ' << f.front() << '\n';
            m.pop_front(); f.pop_front();
             }
       }

    /* Output */
    cout << (n - (m.size() + f.size())) / 2 << ' ' << m.size() << ' ' << f.size();

    /* Return */
    return 0;
}
```

〈헤더파일〉

#. STL Deque, String을 이용하기 위하여 각각의 헤더파일을 선언하였다.

〈Faster〉

#. 싱글 쓰레드 환경에서, printf와 scanf 함수가 이용하는 stdio.h 버퍼와 cin과 cout이 이용하는 iostream의 버퍼의 동기화를 해제함으로서, 입출력 속도를 빠르게 하였다.
ios::sync_with_stdio(0)은 stdio.h와 iostream의 동기화를 해제하겠다는 뜻이다.
cin.tie(0) 및 cout.tie(0)의 경우, cin과 cout은 서로 연결되어 있어 cin을 쓰면 출력 버퍼를 비우고 입력이 발생한다. 이러한 flush(버퍼를 비우는 과정) 과정도 시간이 들기 때문에, cin과 cout의 상호 연결을 끊어주기 위하여 해당 코드를 삽입하는 것이다.

〈Input〉

#. 남자 줄 deque과 여자 줄 deque을 각각 선언한 후, n값을 입력값으로 초기화 한다.

〈Calculation〉

〈Init〉 〈Input〉

#. 시간, 이름, 성별, FR 값을 입력받는다.

〈Deque Insertion〉

#. 성별 및 FR값에 따라, name을 각각의 deque에 삽입한다.

〈Matching〉

#. Deque m과 Deque f가 모두 비어있지 않으면, Matching을 진행한다. 매칭이 완료되면 해당 원소들을 Deque에서 제거한다.

〈Output〉

#. 출력 양식에 알맞은 값을 출력한다.

〈Return〉

#. Main함수에서 값을 반환하고 종료한다.

〈2. 큐 - 정렬 가능 여부 확인 프로그램〉

```cpp
/* 큐 - 정렬 가능 여부 확인 프로그램 */
#include <iostream>
#include <stack>
#include <queue>
using namespace std;

int main(int argc, char* argv[]) {
    /* Faster */
    ios::sync_with_stdio(0);
    cin.tie(0); cout.tie(0);

    /* Init & Input */
    int n; cin >> n;    //n=number
    char s; cin >> s;   //s=sort

    stack<int> stack;
    queue<int> seq;
    for (int i = 0; i < n; i++) {
        int tmp; cin >> tmp;
        seq.push(tmp);
       }

    /* Calculation */
    int i; s == 'a' ? i = 1 : i = n; //asc인 경우 i = 1, desc인 경우 i = n
    while(!seq.empty()) {
        if (stack.empty()) {
            stack.push(seq.front());
            seq.pop();
             }
        while (!seq.empty() ? stack.top() != i : false) {
            stack.push(seq.front());
            seq.pop();
             }
        while (!stack.empty() ? stack.top() == i : false) {
            stack.pop();
            s == 'a' ? i++ : i--;
             }
       }

    /* Output */
    if (stack.empty())
        cout << "Yes";
    else
        cout << "No";

    /* Return */
    return 0;
}
```

〈헤더파일〉

#. stack과 queue를 사용하기 위하여 각각의 헤더파일을 선언하였다.

〈Faster〉

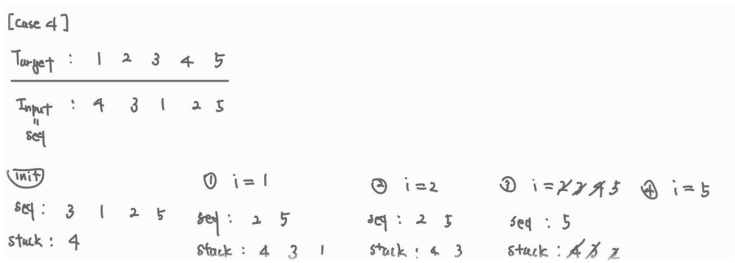
#. 싱글 쓰레드 환경에서, printf와 scanf 함수가 이용하는 stdio.h 버퍼와 cin과 cout이 이용하는 iostream의 버퍼의 동기화를 해제함으로서, 입출력 속도를 빠르게 하였다.
ios::sync_with_stdio(0)은 stdio.h와 iostream의 동기화를 해제하겠다는 뜻이다.
cin.tie(0) 및 cout.tie(0)의 경우, cin과 cout은 서로 연결되어 있어 cin을 쓰면 출력 버퍼를 비우고 입력이 발생한다. 이러한 flush(버퍼를 비우는 과정) 과정도 시간이 들기 때문에, cin과 cout의 상호 연결을 끊어주기 위하여 해당 코드를 삽입하는 것이다.

〈Init & Input〉

#. 입력 값을 입력 받고, stack과 queue를 선언한다. queue에 push되는 값은 입력으로 주어진 수열(sequence)이다.

〈Calculation〉

#. 주어진 알고리즘에 따라 계산을 수행한다. (알고리즘은 대략 위와 같은 방식으로 진행된다.)



〈Output〉

#. 값을 출력한다.

〈Return〉

#. Main함수에서 값을 반환하고 종료한다.

〈3. 큐 - BFS for Maze〉

```cpp
/* BFS for Maze */
#include <iostream>
#include <queue>
#include <vector>
#include <string>
using namespace std;

int main(int argc, char* argv[]) {
    /* Faster */
    ios::sync_with_stdio(0);
    std::cin.tie(0);
    std::cout.tie(0);

    /* Input & Init */
    int n, m; cin >> n >> m;

    int dist[50][50] = {};
    bool visit[50][50] = {};
    char maze[50][50] = {};
    for (int i = 0; i < n; i++)
        cin >> maze[i];

    pair<int, int> T;
    queue<pair<int, int>> q;    // <col, row>

    /* BFS for Maze */
    // find S & T
    for (int x = 0; x < n; x++)
        for (int y = 0; y < m; y++) {
            if (maze[x][y] == 'S') {
                visit[x][y] = true;
                q.push(make_pair(x, y));
                   }
            else if (maze[x][y] == 'T') {
                T.first = x;
                T.second = y;
                   }
                }

    // BFS
    while (!q.empty()) {
        /* Init */
        int x = q.front().first;
        int y = q.front().second;
        q.pop();

        /* Up */
        if (x - 1 >= 0 ?
            (maze[x - 1][y] != '1') && (visit[x - 1][y] == false) : false) {
            dist[x - 1][y] = dist[x][y] + 1;
            visit[x - 1][y] = true;
            q.push(make_pair(x - 1, y));
             }
        /* Down */
        if (x + 1 <= n - 1 ?
            (maze[x + 1][y] != '1') && (visit[x + 1][y] == false) : false) {
            dist[x + 1][y] = dist[x][y] + 1;
            visit[x + 1][y] = true;
            q.push(make_pair(x + 1, y));
             }
        /* Left */
        if (y - 1 >= 0 ?
            (maze[x][y - 1] != '1') && (visit[x][y - 1] == false) : false) {
            dist[x][y - 1] = dist[x][y] + 1;
            visit[x][y - 1] = true;
            q.push(make_pair(x, y - 1));
             }
        /* Right */
        if (y + 1 <= m - 1 ?
            (maze[x][y + 1] != '1') && (visit[x][y + 1] == false) : false) {
            dist[x][y + 1] = dist[x][y] + 1;
            visit[x][y + 1] = true;
            q.push(make_pair(x, y + 1));
             }

        /* Debug */
        //std::cout << '\n';
        //for (int i = 0; i < n; i++) {
        //  for (int j = 0; j < m; j++) {
        //      std::cout << maze[i][j];
        //  }
        //  std::cout << ' ';
        //  for (int j = 0; j < m; j++) {
        //      std::cout << visit[i][j];
        //  }
        //  std::cout << ' ';
        //  for (int j = 0; j < m; j++) {
        //      std::cout << dist[i][j] + 1;
        //  }
        //  std::cout << '\n';
        //}
        //queue<pair<int, int>> _q; _q = q;
        //std::cout << "<QUEUE>" << '\n';
        //while (!_q.empty()) {
        //  std::cout << '(' << _q.front().first << ' ' << _q.front().second << ')' << '\n';
        //  _q.pop();
        //}

       }

    /* Output */
    std::cout << ((dist[T.first][T.second] != 0) ? (dist[T.first][T.second] + 1) : 0);

    /* Return */
    return 0;
}
```

〈헤더파일〉

#. STL queue, pair를 이용하기 위하여 각각의 헤더파일을 선언하였다. (string 헤더파일은 굳이 선언할 필요가 없었다.)

〈Faster〉

#. 싱글 쓰레드 환경에서, printf와 scanf 함수가 이용하는 stdio.h 버퍼와 cin과 cout이 이용하는 iostream의 버퍼의 동기화를 해제함으로서, 입출력 속도를 빠르게 하였다.
ios::sync_with_stdio(0)은 stdio.h와 iostream의 동기화를 해제하겠다는 뜻이다.
cin.tie(0) 및 cout.tie(0)의 경우, cin과 cout은 서로 연결되어 있어 cin을 쓰면 출력 버퍼를 비우고 입력이 발생한다. 이러한 flush(버퍼를 비우는 과정) 과정도 시간이 들기 때문에, cin과 cout의 상호 연결을 끊어주기 위하여 해당 코드를 삽입하는 것이다.

〈Input & Init〉

#. Input으로 들어오는 값인 n, m을 초기화 하였다.
#. dist[][]는 출발지로 부터의 거리를 저장하기 위한 2차원 배열.
#. visit[][]은 방문여부를 체크하기 위한 2차원 배열.
#. maze[][]는 미로를 저장하기 위한 2차원 배열. 이후, Input값으로 초기화를 진행한다.
#. pair〈int, int〉는 도착지 T의 값을 저장하기 위한 pair 변수이다.
#. queue〈pair〈int, int〉〉는 DFS간 탐색 좌표를 저장하기 위한 queue이다.

〈BFS for Maze〉

〈find S & T〉

#. 출발지 S의 좌표를 찾아 queue에 push한다.
#. 도착지 T의 좌표를 찾아 pair에 값을 대입한다.

〈BFS〉

#. BFS를 통해 미로 탐색을 진행한다. 탐색 방향은 상하좌우이며, 해당 좌표들을 큐에 push하기 전에 dist[][]와 visit[][] 값을 업데이트 해준다. 알고리즘 로직은 다음 페이지에 서술하였다.

〈Output〉

#. 출력 양식에 알맞은 값을 출력한다. 탐색이 불가한 경우 T의 값은 0이 된다. 따라서, 이 경우의 처리를 위해 삼항연산자를 사용하였다.

〈Return〉

#. Main함수에서 값을 반환하고 종료한다.

<BFS for Maze 알고리즘 로직>
4 6
1S1000
101010
100010
11100T

1S1000 010000 111111
101010 010000 121111
100010 000000 111111
11100T 000000 111111
<QUEUE>
(1 1)

1S1000 010000 111111
101010 010000 121111
100010 010000 131111
11100T 000000 111111
<QUEUE>
(2 1)

1S1000 010000 111111
101010 010000 121111
100010 011000 134111
11100T 000000 111111
<QUEUE>
(2 2)

1S1000 010000 111111
101010 010000 121111
100010 011100 134511
11100T 000000 111111
<QUEUE>
(2 3)

1S1000 010000 111111
101010 010100 121611
100010 011100 134511
11100T 000100 111611
<QUEUE>
(1 3)
(3 3)

1S1000 010100 111711
101010 010100 121611
100010 011100 134511
11100T 000100 111611
<QUEUE>
(3 3)
(0 3)

1S1000 010100 111711
101010 010100 121611
100010 011100 134511
11100T 000110 111671
<QUEUE>
(0 3)
(3 4)

1S1000 010110 111781
101010 010100 121611
100010 011100 134511
11100T 000110 111671
<QUEUE>
(3 4)
(0 4)

1S1000 010110 111781
101010 010100 121611
100010 011100 134511
11100T 000111 111678
<QUEUE>
(0 4)
(3 5)

1S1000 010111 111789
101010 010100 121611
100010 011100 134511
11100T 000111 111678
<QUEUE>
(3 5)
(0 5)

1S1000 010111 111789
101010 010100 121611
100010 011101 134519
11100T 000111 111678
<QUEUE>
(0 5)
(2 5)

1S1000 010111 111789
101010 010101 1216110
100010 011101 134519
11100T 000111 111678
<QUEUE>
(2 5)
(1 5)

1S1000 010111 111789
101010 010101 1216110
100010 011101 134519
11100T 000111 111678
<QUEUE>
(1 5)

1S1000 010111 111789
101010 010101 1216110
100010 011101 134519
11100T 000111 111678
<QUEUE>

8