서울시립대학교 소모임 알

컴퓨터과학부 김희중, 조용민

2주차 해설

명령 프롬프트

1032

https://www.acmicpc.net/problem/1032

- 저는 단순하게 string 써서 했습니다.

```
3
config.sys
config.inf
configures
```

```
config????
```

1032

https://www.acmicpc.net/problem/1032

- 단순하게 Arr에 string들을 모아놓고 Length만큼 돌면서 다른 값이 나올 때 종료;

```
int num;

scanf("%d", &num);

vector< string > arr;

while (num--){

    char input_str[51];

    scanf("%s", input_str);

    arr.push_back(input_str);
}
```

```
string result = "";

for (int i = 0; i< arr[0].size(); i++){
    char temp = NULL;

    for (int j = 0; j < arr.size(); j++){
        if (temp == NULL){
            temp = arr[j][i];
        }
        else{
            if (temp != arr[j][i]){
                temp = '?';
                break;
            }
        }
    }

    result += temp;
}

printf("%s\n", result.c_str());
```

1032

https://www.acmicpc.net/problem/1032

- C++ Code(github) : https://github.com/OfficialDominyellow/AlgorithmByDominyellow/blob/master/BackjoonOnlineJudge/1032.cc

- (앞선 풀이는 조용민 학생의 풀이입니다. 위의 코드와 해결 방법이 다를 수 있습니다.)

개미

10158

https://www.acmicpc.net/problem/10158

- 이 문제의 input t의 범위는 1 <= t <= 200,000,000이다.
- 한마디로 1초마다 개미를 움직이며 풀려고 하면 바로 시간초과
- 보통 이런 경우 '벽'에 부딛혔을 때를 기준으로 생각.

10158

https://www.acmicpc.net/problem/10158

- 조금만 더 생각해보면 이런 류의 문제들은 2차원처럼 보이지만(x, y가 서로 영향을 준다고 생각할 수도 있지만) x, y는 서로 독립되게 움직인다.

- 한번쯤은 2차원 평면 문제에서 x, y를 따로 움직일 수 있는지 생각하자

10158

https://www.acmicpc.net/problem/10158

- X, Y의 경우를 각각 잘~ 계산하면 되겠습니다.

```
int main(){

    int w, h;

    scanf("%d %d", &w, &h);

    int nowx, nowy;

    scanf("%d %d", &nowx, &nowy);

    int num;

    scanf("%d", &num);

    int movex = num % (2 * w);
    int movey = num % (2 * h);

    if (movex + nowx <= w){
        nowx = (movex + nowx) % (2 * w);
    }
    else if (movex + nowx > w && movex + nowx < 2*w){
        nowx = 2*w-( nowx + movex );
    }
    else if (movex + nowx >=2*w){
        nowx = (movex + nowx) % (2 * w);
    }


    if (movey + nowy <= h){
        nowy = (movey + nowy) % (2 * h);
    }
    else if (movey + nowy > h && movey + nowy  < 2 * h){
        nowy = 2*h-( nowy + movey);
    }
    else if (movey + nowy >= 2 * h){
        nowy = (movey + nowy) % (2 * h);
    }


    printf("%d %d\n", nowx, nowy);
}
```

10158

https://www.acmicpc.net/problem/10158

- C++ Code(github) : https://github.com/OfficialDominyellow/AlgorithmByDominyellow/blob/master/BackjoonOnlineJudge/10158.cc

- (앞선 풀이는 조용민 학생의 풀이입니다. 위의 코드와 해결 방법이 다를 수 있습니다.)

네 수

10824

https://www.acmicpc.net/problem/10824

- String 라이브러리를 이용하면 풀 수 있다.

```
int main(){

    int a, b, c, d;


    scanf("%d %d %d %d", &a, &b, &c, &d);

    string newa, newb;

    newa = to_string(a) + to_string(b);
    newb = to_string(c) + to_string(d);



    printf("%lld\n", stoll(newa) + stoll(newb));
}
```

10824

https://www.acmicpc.net/problem/10824

- C++ Code(github) : https://github.com/OfficialDominyellow/AlgorithmByDominyellow/blob/master/BackjoonOnlineJudge/10824.cc

- (앞선 풀이는 조용민 학생의 풀이입니다. 위의 코드와 해결 방법이 다를 수 있습니다.)

쉽게 푸는 문제

1292

https://www.acmicpc.net/problem/1292

- 인풋의 개수가 1000밖에 안되어서 문제에 쓰인 그대로 배열을 만들었습니다.

```
#include<stdio.h>
#include<vector>
using namespace std;

int main(){

    int start, end;

    scanf("%d %d", &start, &end);


    vector<int> arr;

    for (int i = 1; i <= 45; i++){

        for (int j = 1; j <= i; j++){
            arr.push_back(i);
        }
    }

    int sum = 0;

    for (int i = start-1; i <= end-1; i++){
        sum += arr[i];
    }

    printf("%d\n", sum);

}
```

1292

https://www.acmicpc.net/problem/1292

- C++ Code(github) : https://github.com/OfficialDominyellow/AlgorithmByDominyellow/blob/master/BackjoonOnlineJudge/1292.cc

- (앞선 풀이는 조용민 학생의 풀이입니다. 위의 코드와 해결 방법이 다를 수 있습니다.)

조세퍼스 문제

1158

https://www.acmicpc.net/problem/1158

- 저희 저번시간에 큐 배웠죠 그거 쓰면 됩니다.
- 죽은 사람은 다시 안넣고 계속 큐가 빌때까지 돌립니다.

```
int main(){

    int N, M;

    scanf("%d %d", &N, &M);

    queue<int> q;

    for (int i = 0; i < N; i++){
        q.push(i+1);
    }

    int count = 0;

    vector<int> result;
```

```
while (!q.empty()){

    if (count == M-1){

        result.push_back(q.front());
        q.pop();
        count = 0;
    }
    else{

        int temp = q.front();
        q.pop();
        q.push(temp);

        count++;
    }
}
```

```
printf("<");
for (int i = 0; i < result.size(); i++){

    if (i == result.size() - 1){
        printf("%d>", result[i]);
    }
    else{
        printf("%d, ", result[i]);
    }
}
printf("\n");

}
```

1158

https://www.acmicpc.net/problem/1158

- C++ Code(github) : https://github.com/OfficialDominyellow/AlgorithmByDominyellow/blob/master/BackjoonOnlineJudge/1158.cc

- (앞선 풀이는 조용민 학생의 풀이입니다. 위의 코드와 해결 방법이 다를 수 있습니다.)

경로 찾기

11403

https://www.acmicpc.net/problem/11403

- 저희 저번시간에 BFS/DFS 배웠죠? 그거 쓰면 됩니다.

```
int main(){

    int num;

    scanf("%d", &num);

    map.resize(num);
    result.resize(num);
    for (int i = 0; i < num; i++){

        for (int j = 0; j < num; j++){
            int input;
            scanf("%d", &input);

            map[i].push_back(input);
        }
    }

    for (int i = 0; i < num; i++){

        vector< int> visited;

        visited.resize(num, 0);
        //visited[i] = 1;

        queue<int> q;

        q.push(i);

        while (!q.empty()){

            int temp;
            temp = q.front();
            q.pop();

            for (int j = 0; j < num; j++){
                if (!visited[j] && map[temp][j] == 1){
                    q.push(j);
                    visited[j] = 1;

                }
            }
        }

        for (int k = 0; k < visited.size(); k++){
            result[i].push_back(visited[k]);
        }

    }



    for (int i = 0; i < num; i++){
        for (int j = 0; j < num; j++){
            printf("%d ", result[i][j]);
        }
        printf("\n");
    }


}
```

11403

https://www.acmicpc.net/problem/11403

- C++ Code(github) : https://github.com/OfficialDominyellow/AlgorithmByDominyellow/blob/master/BackjoonOnlineJudge/11403.cc

- (앞선 풀이는 조용민 학생의 풀이입니다. 위의 코드와 해결 방법이 다를 수 있습니다.)

저울

10159

https://www.acmicpc.net/problem/10159

- 방향성이 있는 그래프에 관련된 문제라는 것을 알 수 있다.
- 1번 정점 ~ N번 정점을 시작점으로 dfs를 실행하여 visited배열에 방문했던 정점을 true로 바꾼다.
- 1번 정점 ~ N번 정점을 시작점으로 역 방향 dfs를 실행하여 r_visited배열에 방문했던 정점을 true로 바꾼다.
- Visited과 r_visited 배열에서 true로 체크된 개수를 모두 더하면 무게를 알 수 있는 물건의 개수를 구할 수 있다.

10159

https://www.acmicpc.net/problem/10159

- 1번 정점 ~ N번 정점을 시작점으로 역 방향 dfs를 실행하여 r_visited배열에 방문했던 정점을 true로 바꾼다.
- 저울 문제에서 1 -> 3 간선의 의미는 1의 무게가 3보다 크다는 뜻이다. 반대로 3의 무게가 1보다 가볍다는 의미이다.
- 즉, dfs를 순 방향으로 돌면 정점보다 가벼운 물건들을 모두 탐색할 수 있고, 역 방향으로 돌면 정점보다 무거운 물건들을 탐색할 수 있다.

10159

https://www.acmicpc.net/problem/10159

- C++ Code(github) : https://github.com/OfficialDominyellow/AlgorithmByDominyellow/blob/master/BackjoonOnlineJudge/10159.cc

미로탐색

2178

https://www.acmicpc.net/problem/2178

- BFS를 통해서 거리를 재는 방식으로 문제를 해결한다.
  BFS에서 상당히 자주 나오는 방식! 2차원 배열에 숫자를 채워 넣는다.

1	0	1	1	1	1
1	0	1	0	1	0
1	0	1	0	1	1
1	1	1	0	1	1

1	0	9	10	11	12
2	0	8	0	12	0
3	0	7	0	13	14
4	5	6	0	14	15

2178

https://www.acmicpc.net/problem/2178

- 일단 출발점은 1로 초기화한다.
- BFS를 이용한다.
- BFS 탐색에서 인접한 4 방향을 확인하고 이동할 수 있다면 현재 시간의 +1을 하여 2차원 배열을 채우고, 큐에 push 한다.
- 큐가 비어있을 때 까지 BFS를 반복하면 2차원 배열에 걸리는 시간이 채워져있다.

2178

https://www.acmicpc.net/problem/2178

- 인접한 4 방향 (상, 하, 좌, 우)를 탐색할 때 크게 2가지의 코딩 스타일이 있다.
- 1. 때려 박기
- 2. for문 이용하기
- 어떤 방식이 옳다고 말할 수는 없지만 알고리즘 잘 하는 사람들은 1번처럼 코딩하지 않는 것 같다.

2178

https://www.acmicpc.net/problem/2178


```
while(!q.empty()){
    p = q.front();
    q.pop();

    x = p.x;
    y = p.y;
    t = p.t;

    if(graph[x+1][y] == -1){
      graph[x+1][y] = t+1;
      POINT t_p = {x+1, y, t+1};
      q.push(t_p);
    }
    if(graph[x-1][y] == -1){
      graph[x-1][y] = t+1;
      POINT t_p = {x-1, y, t+1};
      q.push(t_p);
    }
    if(graph[x][y+1] == -1){
      graph[x][y+1] = t+1;
      POINT t_p = {x, y+1, t+1};
      q.push(t_p);
    }
    if(graph[x][y-1] == -1){
      graph[x][y-1] = t+1;
      POINT t_p = {x, y-1, t+1};
      q.push(t_p);
    }
  }
```

1. 때려박기

```
//전역 변수
int x_move[4] = {0, 1, 0, -1};
int y_move[4] = {1, 0, -1, 0};

…

while(!q.empty()){
    p = q.front();
    q.pop();

    x = p.x;
    y = p.y;
    t = p.t;

    for(int i=0; i<4; i++){
      int nx = x + x_move[i];
      int ny = y + y_move[i];

      if(graph[nx][ny] == -1){
        graph[nx][ny] = t+1;
        POINT t_p = {nx, ny, t+1};
        q.push(t_p);
      }
    }
  }
```

2. for문 이용하기

2178

https://www.acmicpc.net/problem/2178

- C++ Code(github) : https://github.com/OfficialDominyellow/AlgorithmByDominyellow/blob/master/BackjoonOnlineJudge/2178.cc

촌수계산

2644

https://www.acmicpc.net/problem/2644

- 전형 적인 무 방향 그래프의 탐색 문제
- DFS 혹은 BFS를 이용하여 정점간의 최단거리를 구한다.

2644

https://www.acmicpc.net/problem/2644

- C++ Code(github) : https://github.com/OfficialDominyellow/AlgorithmByDominyellow/blob/master/BackjoonOnlineJudge/2644.cc

나이트의 이동

7562

https://www.acmicpc.net/problem/7562

- 전형적인 BFS의 최단거리 문제
- 주의할 점! : 8방향의 이동
- 미로탐색 문제에서 다음 칸으로의 이동 가능 여부를 파악할 때 '때려박기'와 'for문 이용하기' 2가지의 방법이 있다고 설명했다.
- 8방향의 이동을 하기 때문에 for문을 이용하는 방식이 코드를 비교적 매우 짧고 명확하게 할 수 있다.

7562

https://www.acmicpc.net/problem/7562

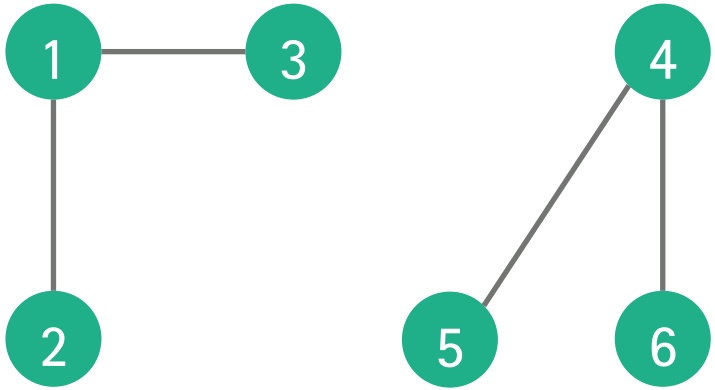
- C++ Code(github) : https://github.com/OfficialDominyellow/AlgorithmByDominyellow/blob/master/BackjoonOnlineJudge/7562.cc

연결 요소의 개수

11724

https://www.acmicpc.net/problem/11724

- 그래프는 항상 연결 그래프가 아닐 수도 있다. = 아래 그림과 같이 나누어져 있을 수도 있다.



11724

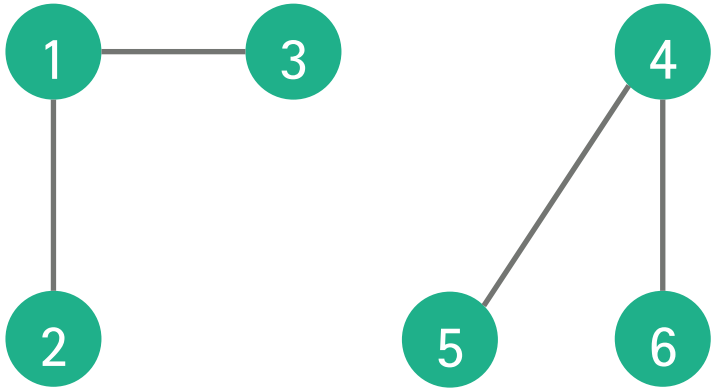
https://www.acmicpc.net/problem/11724

- 이렇게 나누어진 각각의 그래프를 연결 요소라고 한다.
- 연결 요소에 속한 모든 정점을 연결하는 경로가 있어야 한다.
- 또, 다른 연결 요소에 속한 정점과 연결하는 경로가 있어선 안된다.

11724

https://www.acmicpc.net/problem/11724

- 아래 그래프는 2개의 연결 요소로 이루어져 있다.
- BFS, DFS 탐색을 이용해서 연결 요소를 구할 수 있다.



11724

https://www.acmicpc.net/problem/11724

- 탐색이 되지 않은 정점을 찾는 과정을 계속 반복해야 한다.
- 1. visited 배열에서 탐색이 되지 않은 정점을 찾는다.
  탐색되지 않은 정점을 찾았다면 연결요소의 개수를 1 증가시킨다. 찾지 못했다면 연결요소를 모두 찾은 것이다.
- 2. 탐색 되지 않은 정점을 시작점으로 BFS, DFS 탐색을 통해 연결된 정점을 방문하고 visited 배열에 방문 표시를 한다.
- 1, 2번 과정을 반복하면 연결요소의 개수를 구할 수 있다.

11724

https://www.acmicpc.net/problem/11724

- C++ Code(github) : https://github.com/OfficialDominyellow/AlgorithmByDominyellow/blob/master/BackjoonOnlineJudge/11724.cc

유기농 배추

1012

https://www.acmicpc.net/problem/1012

- 앞의 연결 요소의 개수 문제와 완전히 같은 문제라는 것을 알 수 있다.
- 다른 점은 인접 배열의 그래프 형태에서 2차원 배열 필드형태로 판이 달라졌을 뿐이다.
- 권장 : 같은 유형의 2가지 문제를 해결할 때 한 문제를 BFS로 해결했다면 다른 한 문제는 DFS로 해결할 것

1012

https://www.acmicpc.net/problem/1012

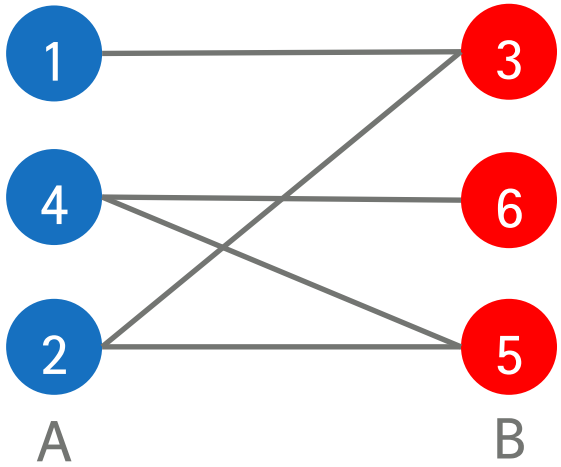
- C++ Code(github) : https://github.com/OfficialDominyellow/AlgorithmByDominyellow/blob/master/BackjoonOnlineJudge/1012.cc

이분 그래프

1707

https://www.acmicpc.net/problem/1707

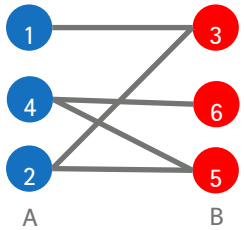
- 그래프를 다음과 같이 A, B로 나눌 수 있으면 이분 그래프(Bipartite Graph)라고 한다.



1707

https://www.acmicpc.net/problem/1707

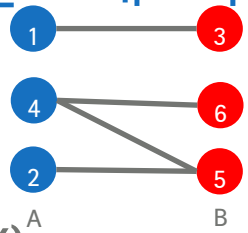
- A에 포함되어 있는 정점 끼리 연결된 간선이 없음
- B에 포함되어 있는 정점기리 연결된 간선이 없음
- 모든 간선의 한 끝 점은 A에, 다른 끝 점은 B에
- 색을 칠해서 구분한다고 생각하면 됨



1707

https://www.acmicpc.net/problem/1707

- BFS, DFS 탐색으로 이분 그래프인지 아닌지 알아낼 수 있다.
- 현재 노드가 파란색이면 그 노드와 연결된 모든 점은 빨간색이다.
- 모든 연결요소가 이분 그래프여야 전체가 이분 그래프라고 할 수 있다. (모든 연결요소에 대해 탐색)
- 노드의 개수가 4천개가 넘는다.
  => 인접리스트를 사용해야 한다! (인접배열X)



1707

https://www.acmicpc.net/problem/1707

- 1 ~ V 까지 모든 노드를 시작점으로 BFS, DFS 탐색을 한다.
- 일단 시작 노드는 파란색(빨간색)으로 초기화 하여 색을 결정한다.
- 현재 노드와 연결된 노드를 자기 자신과 다른 색을 칠하고 탐색을 계속 진행 한다.
- 탐색 중에 이미 방문한 노드이면서 자신과 똑같은 색이 이미 칠해져있는 노드가 나오면 더 볼 것 없이 이분 그래프가 아니다.

1707

https://www.acmicpc.net/problem/1707

- C++ Code(github) : https://github.com/OfficialDominyellow/AlgorithmByDominyellow/blob/master/BackjoonOnlineJudge/1707.cc

목표

5427

https://www.acmicpc.net/problem/5427

- 앞선 미로 탐색 문제의 테크닉이 필요하다.
- 먼저 불이 번져가는 시간을 저장할 int형 이차원 배열이 필요하다.
- 불이 처음부터 있던 위치는 시간을 0으로 하고 좌표를 모두 큐에 집어 넣은 상태로 BFS를 시작한다.
- BFS 탐색이 마무리 되면 좌표에 따라 불이 번지는 시간이 저장된 2차원 배열을 얻어낼 수 있다.

5427

https://www.acmicpc.net/problem/5427

- 그 다음 현수의 위치(@)를 시작점으로 BFS 탐색을 하여 field의 가장자리 밖으로 나갈 수 있는 최단 거리를 구한다.
- 이 때, 인접한 4방향으로 갈수 있는지 여부를 판단할 때 다음과 같은 조건이 만족해야 한다.
- 방문했던 곳 아니다. 벽(#)이 아니다. 불이 없거나 불이 번지는 시간이 현재 시간+1보다 커야 한다.
- 위의 조건을 만족하여 BFS 탐색을 통해 최단거리를 구한다.

5427

https://www.acmicpc.net/problem/5427

- C++ Code(github) : https://github.com/OfficialDominyellow/AlgorithmByDominyellow/blob/master/BackjoonOnlineJudge/5427.cc