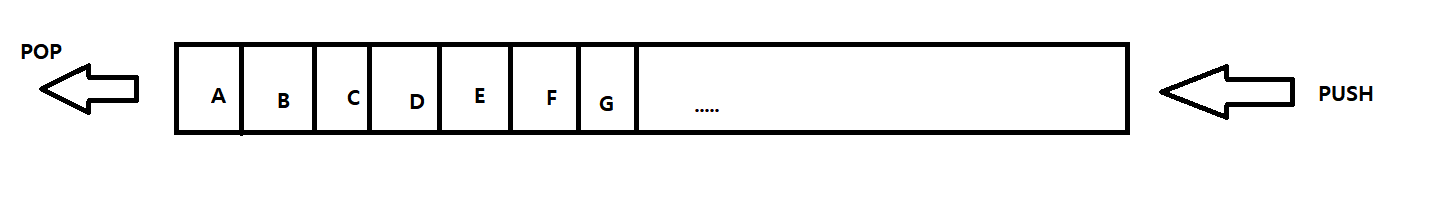
큐(Queue)

-> FIFO(First in First out)

-> 보통 구현은 배열과 링크드 리스트로 구현

-> 하지만 STL로 잘 되어있기 때문에 굳이 구현할 필요는 없다



LIFO(Last In First Out)형태의 자료구조로 위와 같이 ABCDEFG 순서대로 PUSH가 될 경우

POP을 실행하면 ABCDEFG 순서대로 빠져나온다.

stack을 활용한 커서와 키보드 입력 및 삭제 구현

stack은 FIFO(First In First Out)형태로 위의 예시로 들 경우 GFEDCBA 순서대로 POP이 진행된다.

예를 들어 3AB<<>Qd 라는 명령어가 들어오면

3AQ가 출력되어야 함

#include <iostream>  
#include <queue>  
#include <string>  
#include <stack>  
using namespace std;  
​  
int main()  
{  
stack<char> str;  
stack<char> temp;  
string input;  
​  
cin >> input;  
​  
for (int i = 0; i < input.length(); i++)  
{  
​  
if (input[i] == '<')  
{  
if (i != 0 && !str.empty()) {  
temp.push(str.top());  
str.pop();  
}  
continue;  
}  
if (input[i] == '>')  
{  
if (!temp.empty()) {  
str.push(temp.top());  
temp.pop();  
}  
continue;  
​  
}  
if (input[i] == 'd')  
{  
if(!temp.empty())  
temp.pop();  
continue;  
}  
str.push(input[i]);  
​  
}  
​  
if (!str.empty())  
{  
stack<char> save;  
while (!str.empty())  
{  
save.push(str.top());  
str.pop();  
}  
while(!save.empty())  
{  
cout << save.top();  
save.pop();  
}  
}  
​  
if (!temp.empty())  
{  
while (!temp.empty())  
{  
cout << temp.top();  
temp.pop();  
}  
}  
​  
​  
return 0;  
}

G,A,T,R 네 사람이 있을 때, 5번째 지목될 때 마다 죽이고 마지막 한 명 출력하기

#include <iostream>  
#include <queue>  
#include <string>  
#include <vector>  
#include <stack>  
using namespace std;  
​  
int main()  
{  
queue<char> que;  
int cnt = 1;  
​  
que.push('G');  
que.push('A');  
que.push('T');  
que.push('R');  
  
while (1)  
{  
if (que.size() == 1)  
break;  
if(cnt!=5)  
que.push(que.front());  
if (cnt == 5)  
cnt = 0;  
que.pop();  
cnt++;  
}  
​  
while (!que.empty())  
{  
cout << que.front() << endl;  
que.pop();  
}  
​  
return 0;  
}

#include <iostream>  
#include <string>  
using namespace std;  
​  
string str = "ABCDEFGH";  
int used[100] = { 0 };  
​  
int nextPer(int idx)  
{  
   for (int i = idx + 1; i < str.length() \* 2; i++) {  
       int tx = i % str.length();  
       if (used[tx] == 1) continue;  
       return tx;  
  }  
   return 0;  
}  
​  
int selectDie(int idx)  
{  
   //다섯번째 사람을 선택  
​  
   for (int i = 0; i < 5; i++) {  
       idx = nextPer(idx);  
  }  
​  
   return idx;  
}  
​  
int main()  
{  
   int cnt = str.length();  
   int idx = -1;  
​  
   while (cnt != 1) {  
       idx = selectDie(idx);  
       used[idx] = 1;  
       cnt--;  
  }  
​  
   for (int i = 0; i < str.length(); i++) {  
       if (used[i] == 0) {  
           cout << str[i] << endl;  
           break;  
      }  
  }  
     
​  
   return 0;  
}

2020카카오 겨울 인턴십 인형뽑기

#include <iostream>  
#include <vector>  
#include <stack>  
using namespace std;  
​  
stack<int> mapst[6];  
stack<int> result;  
stack<int> temp;  
​  
void go(vector<vector<int>> map, vector<int> moves)  
{  
   //go 함수를 완성  
   //출력결과 : (4)4 2 4   
               //터진개수와 쌓아둔 개수를 출력  
​  
   //1. 2차원배열 map을 mapst에 넣기  
   for (int x = 0; x < 5; x++) {  
       for (int y = 4; y >= 0; y--) {  
​  
           if (map[y][x] == 0) break;  
           mapst[x + 1].push(map[y][x]);  
      }  
  }  
​  
​  
   //2. move 하나씩 수행 result에 push  
​  
   int cnt = 0;  
   for (int i = 0; i < moves.size(); i++) {  
​  
       int idx = moves[i];  
       if (mapst[idx].size() == 0) continue;  
       int ret = mapst[idx].top();  
       mapst[idx].pop();  
​  
       if (result.size() > 0 && result.top() == ret) {  
           result.pop();  
           cnt += 2;  
      }  
       else {  
           result.push(ret);  
      }  
  }  
​  
   while (!result.empty())  
  {  
       temp.push(result.top());  
       result.pop();  
  }  
​  
   cout << cnt << " ";  
   while (!temp.empty()) {  
       cout << temp.top();  
       temp.pop();  
  }  
}  
​  
int main()  
{  
   vector<int> moves = { 1,5,3,5,1,2,1,4 };  
   vector<vector<int>> map = {  
      {0,0,0,0,0},  
      {0,0,1,0,3},  
      {0,2,5,0,1},  
      {4,2,4,4,2},  
      {3,5,1,3,1},  
  };  
​  
   go(map, moves);  
​  
   return 0;  
}

강사님 코드

#include <iostream>  
#include <vector>  
#include <stack>  
using namespace std;  
​  
void go(vector<vector<int>> map, vector<int> moves)  
{  
//go 함수를 완성  
//출력결과 : (4)4 2 4   
//터진개수와 쌓아둔 개수를 출력  
stack<int> box;  
int idx = 0;  
int cnt = 0;  
  
for (int i = 0; i < moves.size(); i++)  
{  
for (int j = 0; j < map.size(); j++)  
{  
int num = map[j][moves[i] - 1];  
map[j][moves[i] - 1] = 0;  
​  
if (num != 0)  
{  
if (box.empty()) {  
box.push(num);  
break;  
}  
else  
{  
if (box.top() == num)  
{  
cnt += 2;  
box.pop();  
break;  
}  
else {  
box.push(num);  
break;  
}  
}  
}  
}  
}  
​  
stack<int> ans;  
while (!box.empty())  
{  
ans.push(box.top());  
box.pop();  
}  
cout << "(" << cnt << ")";  
while (!ans.empty())  
{  
cout << ans.top() << " ";  
ans.pop();  
}  
}  
​  
int main()  
{  
vector<int> moves = { 1,5,3,5,1,2,1,4 };  
vector<vector<int>> map = {  
{0,0,0,0,0},  
{0,0,1,0,3},  
{0,2,5,0,1},  
{4,2,4,4,2},  
{3,5,1,3,1},  
};  
​  
go(map, moves);  
​  
return 0;  
}

내 코드

링크드 리스트

#include <iostream>  
using namespace std;  
​  
struct Node {  
   int n;  
   Node \*next;  
};  
​  
Node \*head, \*last;  
​  
void addNode(int n)  
{  
   if (head == NULL) {  
       head = new Node();  
       head->n = n;  
       last = head;  
       return;  
  }  
​  
   last->next = new Node();  
   last = last->next;  
   last->n = n;  
}  
​  
int main()  
{  
   int v[5] = { 4, 8, 7, 6, 9 };  
​  
   addNode(4);  
   addNode(8);  
   addNode(7);  
   addNode(6);  
   addNode(9);  
​  
   for (int i = 0; i < 5; i++) {  
       cout << v[i];  
  }  
   cout << endl;  
   for (Node \*p = head; p != NULL; p = p->next) {  
       cout << p->n;  
  }  
​  
   return 0;  
}

그래프 하드코딩

1. 인접 행렬
2. 인접 리스트

그래프 탐색

1. DFS
2. BFS

그래프 안의 값이 순차적 고유번호 임이 확정 되었을 때 => Graph의 number

그래프 안의 값이 순차적이지 않고 중복된 값이 존재할 때 => Graph의 value

int map[4][4]={  
{0,1,0,0},  
{0,0,1,1},  
{0,1,0,0},  
{0,0,1,0}  
}; //인접 행렬로 초기화

int map[5][5]={  
{0,0,0,0,0},  
{1,0,0,0,0},  
{0,1,0,1,0},  
{0,1,0,0,1},  
{0,0,0,0,0}  
};//인접 리스트 초기화  
int list[5]={0,2,3,4,3};

#include<iostream>  
using namespace std;  
​  
int graph[5][5]={  
{0,0,0,0,1},  
{1,0,0,0,1},  
{0,0,0,0,1},  
{0,1,1,0,0},  
{0,0,0,1,0}  
};  
char list[5]={'A','B','C','F','D'};  
int dat[26]={};  
int cnt=0;  
int main(){  
for(int i=0;i<5;i++)  
{  
for(int j=0;j<5;j++)  
{  
if(graph[i][j]==1){  
dat[list[j]-65]++;  
}  
}  
}  
​  
int max=-9999999;  
int max\_idx=0;  
​  
for(int i=0;i<26;i++)  
{  
if(dat[i]==0) continue;  
if(max<dat[i])  
{  
max=dat[i];  
max\_idx=i;  
}  
}  
​  
cout<<(char)(max\_idx+65)<<endl;  
   return 0;  
}

트리

그래프에서 양방향이 아닌 단 방향으로 뻗어나온 그래프(Cycle 존재 X)

void run(int now){   
cout<<now<<" ";  
for(int i=0;i<6;i++){  
if(map[now][i]==0) continue;  
run(i);  
}  
}

dfs 연습

#include <iostream>  
#include <vector>  
using namespace std;  
​  
vector<vector<int>> alist(9);  
int val[9] = { 0, 1, 2, 7, 3, 6, 9, 15, 16 };  
​  
void run(int now)  
{  
   cout << val[now] << " ";  
   for (int i = 0; i < alist[now].size(); i++) {  
​  
       int next = alist[now][i];  
       run(next);  
​  
  }  
}  
​  
int main()  
{  
   alist[0] = { 1, 2, 3 };  
   alist[2] = { 4, 5 };  
   alist[3] = { 6, 7, 8 };  
     
   run(3);  
​  
   return 0;  
}

used 배열을 탐색 후 0으로 초기화 시키면 -> 모든 경로를 탐색

초기화 시키지 않으면 -> 모든 노드를 탐색

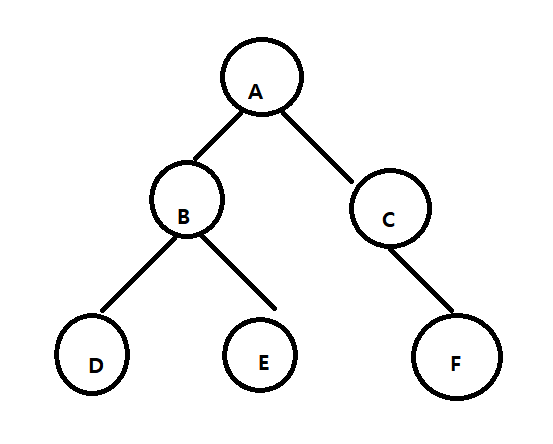
#include <iostream>  
#include <vector>  
#include <queue>  
using namespace std;  
​  
int map[4][4] = {  
{0,1,1,1},  
{0,0,1,1},  
{0,0,0,0},  
{1,0,1,0}  
};  
int used[4] = {};  
int cnt = 0;  
​  
void dfs(int n) {  
if (n == 2) cnt++;  
​  
for (int i = 0; i < 4; i++) {  
if (used[i] == 1) continue;  
if (map[n][i] == 0) continue;  
used[i] = 1;  
dfs(i);  
   used[i] = 0;  
}  
}  
​  
int main()  
{  
dfs(0);  
​  
cout << cnt <<"개"<< endl;  
​  
return 0;  
}

#include <iostream>  
#include <queue>  
using namespace std;  
​  
int map[7][7] = {  
0,1,1,0,0,0,0,  
0,0,1,0,0,0,0,  
0,0,0,1,1,1,0,  
0,1,0,0,0,0,0,  
1,0,0,0,0,1,1,  
0,0,0,1,0,0,0,  
0,0,0,0,0,0,0,  
};  
​  
char name[8] = "AGKRHTV";  
int used[10];  
​  
queue<int> q;  
​  
int main()  
{  
   used[2] = 1;  
   q.push(2);  
​  
   while (!q.empty()) {  
​  
       int now = q.front();  
       q.pop();  
​  
       cout << name[now] << " ";  
​  
       for (int i = 0; i < 7; i++) {  
           if (map[now][i] == 0) continue;  
           if (used[i] == 1) continue;  
           used[i] = 1;  
           q.push(i);  
      }  
  }  
​  
   return 0;  
}

bfs cycle 검사

DFS는 재귀를 생각하면 되고 BFS는 queue를 이용해서 저장된 위치를 탐색하는 기법

DFS는 깊이 우선 탐색이고 BFS는 너비우선 탐색이다



다음과 같은 그래프가 존재할 때

DFS로 탐색할 경우 A -> B -> D -> E -> C -> F 순서로 탐색이 진행된다.

이때 재귀나 stack를 활용해서 자신이 인접해 있는 행렬과 그 행렬이 인접한 행렬을 쭉 들어가면서 탐색을 진행하면 된다.

BFS로 탐색할 경우 A -> B -> C -> D -> E -> F 순서로 탐색이 진행된다.

이때 queue를 활용해서 자신이 인접한 행렬만 쭉 탐색을 진행하면 된다.