트리 -> 단방향, 사이클 존재X

그래프 -> 단,양방향, 사이클 존재O

트리를 이용한 BFS

#include <iostream>  
#include <queue>  
using namespace std;  
​  
char name[]={'A','B','C','D','E'};  
int map[5][5]={  
  {0,1,1,0,0},  
  {0,0,0,1,1},  
  {0,0,0,0,0},  
  {0,0,0,0,0},  
  {0,0,0,0,0}  
};  
int queue[11];  
int head=0,tail=1;  
//DFS => ABDEC  
//BFS => ABCDE -> 지금 내 현재 위치에서 갈 수 있는 곳을 queue에 넣기  
int main(){  
 queue[0]=0;  
 while(head!=tail){  
     int now=queue[head++];  
     cout<<map[now]<<" ";  
       
     for(int i=0;i<5;i++){  
         if(map[now][i]==1){  
             queue[tail++]=i;  
        }  
    }  
}  
   cout<<endl;  
     
   return 0;  
}

queue 사용

#include <iostream>  
#include <queue>  
using namespace std;  
​  
char name[]={'A','B','C','D','E'};  
int map[5][5]={  
  {0,1,1,0,0},  
  {0,0,0,1,1},  
  {0,0,0,0,0},  
  {0,0,0,0,0},  
  {0,0,0,0,0}  
};  
queue<int> que;  
​  
//DFS => ABDEC  
//BFS => ABCDE -> 지금 내 현재 위치에서 갈 수 있는 곳을 queue에 넣기  
int main(){  
 que.push(0);  
 while(!que.empty()){  
     int now=que.front();  
     que.pop();  
     cout<<map[now]<<" ";  
       
     for(int i=0;i<5;i++){  
         if(map[now][i]==1){  
            que.push(i);  
        }  
    }  
}  
   cout<<endl;  
     
   return 0;  
}

인접 리스트를 활용한 bfs

#include <iostream>  
#include <vector>  
#include <queue>  
using namespace std;  
​  
vector<vector<int>> vect(5);  
queue<int> q;  
char name[5]="ABCDE"  
int main(){  
vect[0].push\_back(1);  
   vect[0].push\_back(2);  
   vect[1].push\_back(3);  
   vect[1].push\_back(4);  
     
   q.push(0);  
     
   while(!q.empty()){  
       int now = q.front();  
       q.pop();  
         
       cout<<name[now]<<" ";  
       for(int x=0; x<vect[now].size(); x++)  
      {  
           int next = vect[now][x];  
           q.push(next);  
      }  
       
  }  
     
return 0;  
}

그래프를 1번씩 탐색하는 BFS

그래프 탐색해서 path 출력하기

#include <iostream>  
using namespace std;  
int map[4][4]={  
   0,1,1,1,  
   0,0,0,1,  
   0,0,0,1,  
   1,0,1,0,  
};  
//인접행렬 이용  
struct node{  
   int index;  
   int level;  
   int used[4];  
   char path[4];  
}que[100]={0,0,{1},{'A'}};  
int head=0,tail=1;  
char name[4]="ABCD";  
void bfs(){  
   while(head!=tail){  
       node now = que[head++];  
         
       if(name[now.index]=='D'){  
           cout<<now.path<<endl;  
      }  
       for(int i=0;i<4;i++){  
           if(map[now.index][i]==1 && now.used[i]==0)  
           que[tail]=now;  
           que[tail].index=i;  
           que[tail].level=now.level+1;  
           que[tail].used[i]=1;  
           que[tail].path[now.level+1]=name[i];  
           tail++;  
      }  
  }  
}  
​  
int main(){  
   bfs();  
   return 0;  
}

슬라이딩 윈도우

#include <iostream>  
using namespace std;  
int arr[11]={3,17,4,11,1,5,2,3,4,5,7};  
//연속된 5개의 합을 구하는데, 합이 최대인 곳의 좌표 출력  
int main(){  
     
   int max\_=-99999999;  
   int max\_idx=0;  
   for(int i=0;i<11;i++){  
      if(i+5>10) break;  
       for(int j=0;j<5;j++){  
           sum+=arr[i+j];  
      }  
       if(max\_<sum){  
           max\_=sum;  
           max\_idx=i;  
      }  
  } //= > 시간초과.. O(n^2)  
   cout<<max\_idx<<endl;  
     
   return 0;  
}

O(n)의 속도로 해결하기 위한 방법 -> 슬라이딩 윈도우, 투 포인터

#include <iostream>  
using namespace std;  
int arr[11] = { 3,17,4,11,1,5,2,3,4,5,7 };  
//연속된 5개의 합을 구하는데, 합이 최대인 곳의 좌표 출력  
int main() {  
​  
int max\_ = -99999999;  
int max\_idx = 0;  
int sum = 0;  
​  
for (int i = 0; i < 5; i++) {  
sum += arr[i];  
}  
​  
for (int i = 0; i < 6; i++) {  
if (sum > max\_) {  
max\_ = sum;  
max\_idx = i;  
}  
sum += arr[i + 5];  
sum -= arr[i];  
}  
​  
cout << max\_idx << endl;  
​  
return 0;  
}//O(n)의 속도로 처리 ㄱ

#include<iostream>  
using namespace std;  
int main()  
{  
   int n = 8;//음식 개수  
   int r = 2;//범위 거리  
   int arr\_temp[] = { 65, 65, 81, 66, 65, 65, 69, 69 };  
   int arr[16] = { 0 }; // 0~7 인덱스 원본 8~15 인덱스 까지 복사본  
​  
for (int x = 0; x < 8; x++)  
{  
   arr[x] = arr\_temp[x];  
   arr[x + n] = arr[x];  
}  
​  
// 로테이션 준비  
// 원하는것 은 : 각 음식이 몇개씩 있는가?? 3개 이상이면 땡  
​  
int dat[201] = { 0 };  
int check = 0;  
​  
// 슬라이딩 윈도 해주는데 확인할 구간은  
// 2\*r +1개 를 확인  
​  
for (int x = 0; x < 2 \* r; x++)  
{  
   //2\*r개에 대한 처리  
   dat[arr[x]]++;  
   if (dat[arr[x]] >= 3)  
  {  
       check = 1;  
  }  
}  
​  
for (int x = 0; x < n; x++)  
{  
   //2\*r 개-> 2\*r+1개를 확인  
​  
   dat[arr[x + 2 \* r]]++;

   // 2\*r+1개의 음식에 대한 처리  
​  
   if (dat[arr[x + 2 \* r]] >= 3) {  
       check = 1;  
  }  
   // 다음 구간에 겹치는 음식만 남기기 위해서  
​  
   // 현재 구간의 맨앞 data를 삭제  
​  
   dat[arr[x]]--;  
}  
​  
​  
if (check == 1)cout << "노";  
else cout << "예스";  
​  
return 0;  
}

슬라이딩 윈도우 알고리즘은 비교하고자 하는 배열의 크기를 먼저 그 크기만 큼 저장해 준 후

배열의 크기에서 비교하고자 하는 크기만 큼 뺀 크기를 반복해서 앞이랑 뒤를 더하고 빼가면서 비교하는 알고리즘이다.

통상 O(n^2)의 속도로 진행할 것을 O(n)속도로 줄일 수 있다.