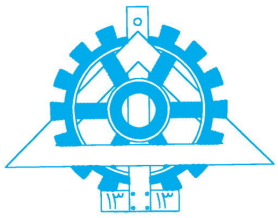
تمرین شماره 5

Graph



دانشکده مهندسی برق و کامپیوتر

استاد: دکتر هشام فیلی

مهلت تحویل:

1400/10/4, ساعت: 23:59

ساختمان های داده و الگوریتم - پاییز 1400

طراح تمرین : **علیرضا آقایی**

سوال ۱ (۱۵ نمره)

DFS:

یال‌های داخل درخت تولید شده:

(1,2) , (2, 5), (5, 6), (6, 3), (1, 4), (7, 9), (9, 8)

Backedges: (3, 1), (9, 7)

Cross edges: (7, 4)

زمان‌های ورود و خروج:

node	Starting time	Finishing time
1	0	11

2	1	8
3	4	5
4	9	10
5	2	7
6	3	6
7	12	17
8	14	15
9	13	16

BFS:

یال‌های داخل درخت:

(1, 2), (1, 4), (1,5), (2, 6), (6, 3), (7, 9), (9, 8)

Backedges: (3, 1)

Crossedges: (2, 5), (7, 4)

* بستگی به ترتیب پیمایش همسایه‌ها ممکن است درخت پیمایش متفاوت باشد و جواب یکتا نیست.

سوال ۲ (۱۵ نمره)

از رأس s داده شده dfs می‌زنیم. هر بار از رأس روی همسایه هایش dfs را روی آن‌هایی که in_path آن‌ها false است اجرا می‌کنیم. زمانی که به رأس d رسیدیم مسیر پیدا شده را چاپ می‌کنیم، سپس in_path رأس d را برابر با false قرار می‌دهیم و backtrack می‌کنیم. اگر هم به بن بست رسیدیم و نمی‌توانستیم ادامه دهیم دوباره backtrack می‌کنیم تا در نهایت تمام مسیرها را به دست بیاوریم.

شبه کد:

```
void dfs(int u, int d, bool in_path[],int path[], int& path_index){
    in_path[u] = true;
    path[path_index] = u;
    path_index++;
    if (u == d) {
        for (int i = 0; i < path_index; i++)
            cout << path[i] << " ";
        cout << endl;
    }
    else {
        for (i : adj[u])
            if (!in_path[i]) dfs(i, d, visited, path, path_index);
```

```
    }
    path_index--;
    in_path[u] = false;
}
```

پیچیدگی زمانی این الگوریتم با توجه به این که بدترین حالت ممکن برای گراف ورودی گراف کامل است،

$O(n!)$ است.

سوال ۳ (۱۵ نمره)

دو رأس یال‌هایی که نیاز به تعمیر دارند را سفید در نظر می‌گیریم. سپس $d[v]$ را برابر با تعداد رئوس سفید در زیر درخت رأس v در نظر می‌گیریم. مقدار $d[v]$ را با استفاده از dfs به این صورت به دست می‌آوریم:

```
calc(v, prev)
{
	d[v] = 0;
	if (white[v])
		dv += 1;
	for all vertices u such that there is the edge (u,v) or (v,u), u != prev:
		calc(u, v);
		d[v] += d[u];

}
```

در نهایت رئوسی را انتخاب می‌کنیم که سفید هستند و مقدار $d[v]$ آن‌ها برابر با یک است. پیچیدگی این الگوریتم مشابه پیمایش dfs برابر O(n + m) است.

اثبات: حال باید کمینه بودن این تعداد را اثبات کنیم. برای این کار از روش معمول اثبات الگوریتم‌های حریصانه (در درس طراحی الگوریتم با این روش حل مساله بیشتر آشنا خواهید شد) استفاده می کنیم. از برهان خلف استفاده می‌کنیم، فرض کنید جوابی بهینه با تعداد راس کمتری وجود داشته باشد. شبیه ترین جواب بهینه به جواب خودمان در نظر می‌گیریم (شبیه ترین یعنی جوابی که بیشترین اشتراک راس را دارد) حال جواب بهینه را در نظر بگیرید، اگر هر یک از راس‌های سفید با $d[v] = 1$ را انتخاب نکنیم، باید حتما یک راس از این زیر درخت (u) انتخاب شود یال این راس به پدرش (که خراب است) را پوشش دهد. حال اگر راس u را از جواب بهینه حذف و راس v را اضافه کنیم باز هم تمام یال‌های خراب پوشش داده شده‌اند اما جوابی شبیه تر به جواب بهینه داریم که این خلاف فرض است.

همچنین واضح است الگوریتم ما همه یال‌های خراب را پوشش می دهد زیرا در زیر درخت آن حتما یک راس سفید با $d[v] = 1$ وجود دارد. پس اثبات شد که جواب ما درست و بهینه است.

سوال ۴ (۱۵ نمره)

گراف داده شده اگر دور فرد داشته باشد، اگر آن را جهت دهی کنیم، حتما دو یال متوالی از آن دور جهت یکسان خواهند داشت و بنابراین حتما یک مسیر جهت‌دار با طول ۲ خواهیم داشت. پس در این حالت ثابت شد که نمی‌توان این کار را انجام داد. اما می‌دانیم در صورتی که گراف داده شده دور فرد نداشته باشد به این معناست که

گراف دو بخشی است. حال کافی است جهت همه‌ی یال‌ها را از بخش ۱ به بخش ۲ بگذاریم، در این صورت هیچ مسیر جهت‌داری با طول ۲ یا بیشتر نخواهیم داشت.

برای انجام این کار نیز از dfs استفاده می‌کنیم و شروع به رنگ‌آمیزی رأس‌ها با دو رنگ سفید و مشکی به صورت یک در میان می‌کنیم. (رنگ راس را به عنوان پارامتر تابع dfs ورودی می دهیم. برای همسایه ها این رنگ را برعکس می کنیم) در صورتی که پس از اجرای الگوریتم dfs، برای یک یال دو سر آن همرنگ بودند به این معناست که گراف دور فرد دارد و نمی‌توان جهت دهی مورد نظر را انجام داد. در غیر این صورت کافی است جهت یال‌ها را از طرف رنگ سفید به سیاه مشخص کنیم. پیچیدگی زمانی این الگوریتم نیز $O(n + m)$ است.

سوال ۵ (۲۰ نمره)

به این صورت عمل می‌کنیم که با dfs تمام رئوس برشی را به دست بیاوریم. رأسی برشی است که یکی از دو ویژگی زیر را داشته باشد:

۱- رأس x ریشه درخت dfs باشد و حداقل دو همسایه داشته باشد.

۲- رأس x ریشه درخت dfs نباشد و به ازای هر همسایه آن مانند v داشته باشد که از هیچ کدام از رئوس داخل زیر درخت آن، هیچ backedge به رئوس پدر رأس x وجود نداشته باشد.

$disc[v]$ را برابر با starting time رأس v در نظر می‌گیریم. همچنین $low[v]$ را کوچکترین مقدار disc بین فرزندان راس v در نظر می‌گیریم (به جز پدر رأس v در درخت dfs، همچنین می توانید ارتفاع را به جای starting

time در نظر بگیرید). حال روی گراف dfs می‌زنیم و به این صورت مقدار low را آپدیت می‌کنیم: (رأس کنونی dfs را x در نظر می‌گیریم و روی فرزند آن مانند u حالت بندی می‌کنیم)

۱- اگر u در گذشته visit نشده باشد: ابتدا dfs را روی u اجرا می‌کنیم و در نهایت:

$$low[x] = min(low[x], low[u])$$

اگر شرط $low[u] >= disc[x]$ برقرار بود آنگاه رأس x برشی است.

۲- اگر u پدر رأس x نبود و در گذشته دیده شده بود:

$$low[x] = min(low[x], disc[u])$$

کد: (شبه کد نیز کفایت می کند)

```
void dfs(vector<int> adj[], int u, bool visited[], int disc[], int low[], int& time, int parent, bool isAP[]) {
    int children = 0;
    visited[u] = true;
    disc[u] = low[u] = ++time;
    for (auto v : adj[u]) {
        if (!visited[v]) {
            children++;
            dfs(adj, v, visited, disc, low, time, u, isAP);
            low[u] = min(low[u], low[v]);
            if (parent != -1 && low[v] >= disc[u])
```

```
                    isAP[u] = true;
            }
            else if (v != parent)
                low[u] = min(low[u], disc[v]);
        }
        if (parent == -1 && children > 1)
            isAP[u] = true;
}
```

سوال ۶ (۲۰ نمره)

از یکی از دو الگوریتم Kosaraju یا Tarjan استفاده می‌کنیم و مؤلفه‌های قویا همبند گراف را به دست می‌آوریم.

حال از هر مؤلفه‌ی قویا همبند رأسی را انتخاب می‌کنیم که کمترین هزینه را داشته باشد.

برای به دست آوردن تعداد جواب های ممکن نیز به این صورت عمل می‌کنیم که تعداد حالت‌های انتخاب کم‌هزینه ترین رأس از هر مؤلفه را به دست آورده و طبق اصل ضرب در ترکیبیات، در هم ضرب می‌کنیم تا به جواب نهایی برسیم.