نمایش گراف‌ها

تعریف

نمایش گراف در واقع همان روش‌های ذخیره‌سازی گراف در کامپیوتر است، به عبارت دیگر با توجه به محدودیت‌هایی که در کامپیوتر داریم، نمی‌توانیم شکل گراف را همان‌طور که در کاغذ می‌کشیم نشان دهیم، یا با گفتن ویژگی‌های تصویری آن را به راحتی ذخیره و شبیه سازی کنیم.
از این‌رو روش‌های مختلفی برای اینکار بوجود آمده است.

شماره‌گذاری

اولین کاری که برای ذخیره سازی گراف نیاز داریم، شماره‌گذاری رئوس است. یعنی به هر رأسی یک شماره نسبت دهیم تا بتوانیم بین آن‌ها قائل شویم. از این‌رو گراف‌های یک شکل، نمایشی متفاوت دارند. چون آرایه‌ها و آدرس‌ها در کامپیوتر از صفر شروع می‌شوند معمولا شماره‌گذاری را از صفر (نه از یک) آغاز می‌کنند.

ماتریس مجاورت

ماتریس مجاورت، در واقع یک جدول دوبعدی از درایه‌ها است که طول سطر و ستون آن هر دو برابر تعداد رأس‌های گراف است. ابتدا رأس‌ها را شماره گذاری می‌کنیم. حال در درایه‌ی سطر iام و ستون jام آن اگر از رأس شماره i به j یال نبود، صفر می‌گذاریم؛ اگر یال بود وزن آن را و اگر گراف وزن‌دار نبود، ۱ می‌گذاریم. همچنین اگر گراف بدون‌جهت باشد، برای قرینه آن هم انجام می‌دهیم یعنی این‌بار همین کار را از سطر j به ستون i انجام می‌دهیم.

در این شیوه ذخیره سازی چون تعداد سطرها و ستون‌ها برابر تعداد رئوس است پس به فضای حافظه‌ای از $O(n^2)$ نیاز داریم که n تعداد رأس‌ها است. اگرچه ممکن است کمی زیاد بنظر بیاید، اما خوبی این شیوه در این است که با $O(1)$ می‌توان از وزن یال بین دو رأس در صورت وجود اطلاع یافت که در شیوه‌های دیگر این ویژگی را نداریم.

پیاده‌سازی

معمولا برای پیاده‌سازی ماتریس مجاورت از یک آرایه دوبعدی استفاده می‌شود. و فرض می‌کنیم که گراف ورودی، یک گراف جهت‌دار و وزن‌دار است.

```
#include <iostream>

using namespace std;

const int MAXN = 1000 + 10
int adj[MAXN][MAXN];  //  ماتریس مجاورت
int main() {
    int n, m;     //  به ترتیب از چپ به راست تعداد رأس‌ها و یال‌ها هستند
    cin >> n >> m;

    for(int i = 0; i < m; i++) {
        int v, u;
        cin >> v >> u >> w;
        adj[--v][--u] = w;     // معمولا شماره رأس‌ها از یک شروع می‌شوند ولی ما از صفر شروع می‌کنیم
    }
}
```

لیست مجاورت

لیست مجاورت، در واقع لیستی است که به ازای هر رأسی لیستی از مجموعه یال‌هایش (یال‌های خروجی در گراف‌های جهت‌دار) را نگه می‌داریم. پس فضای حافظه‌ی ما به تعداد یال‌ها وابسته است؛ با کمی تفکر میتوان دریافت که فضای مصرفی در گراف‌های بی‌جهت و جهت‌دار به ترتیب دوبرابر و برابر تعداد یال‌هاست.

از آنجایی که برای هر رأسی تعداد یال‌های متفاوتی را نگه می‌داریم، می‌توانیم به ازای هر رأسی یک لیست پیوندی بگیریم، اما از طرفی هم میتوان از ابزارهای دیگری نیز استفاده کرد که به صورت سرشکن فضای اضافه‌ای نمی‌گیرند و در عمل کار را ساده‌تر می‌کنند.

با توجه به مطالب گفته شده، مرتبه حافظه‌ای مورد نظر برای اینکار از $O(n+e)$ است که برای گراف های تنک، فضای بهینه و کمی است. اما برای گراف های شلوغ، ماتریس مجاورت بهتر است، چراکه در این حالت با وجود فضای مصرفی‌ای به اندازه ماتریس مجاورت، همجنان برای فهمیدن وجود یال بین دو راس v و u در بدترین حالت باید کل لیست را بگردیم که یعنی از $O(n)$ زمان مصرف می‌کنیم.

پیاده‌سازی

برای پیاده سازی، از $vector$ استفاده کرده‌ایم زیرا درست است که از نظر زمانی و حافظه‌ای بیشتر از لیست پیوندی هزینه دارد اما در عوض کد زدن با آن راحت‌تر است و معمولا این ضریب‌ها در زمان اجرا و حافظه‌ی مصرفی تاثیر بسیار کمی می‌گذارند در نتیجه قابل چشم پوشی‌اند.
کد زیر برای گراف‌های جهت‌دار و وزن‌دار زده شده است، برای گراف های بی‌وزن میتوانید بجای $pair$ از int استفاده کنید.

```
#include <iostream>
#include <vector>

using namespace std;

const int MAXN = 1000 * 100 + 10;

vector <pair<int, int> > adj[MAXN];

int main() {
    int n, m;
    cin >> n >> m;
    for(int i = 0; i < m; i++) {
        int v, u, w;
        cin >> v >> u >> w;
        adj[--v].push_back(pair<int, int>(--u, w));
    }
    return 0;
}
```

کابردها

ذخیره‌سازی گراف‌ها در کامپیوتر