به نام خدا

طراحی الگوریتم

نمونه امتحان دوم

مدت امتحان: ۲ ساعت

توجه

- توصیه میشود قبل از خواندن پاسخ‌ها، سعی کنید سوالات را خودتان حل نمایید.

۱. (۲۰ نمره) فرض کنید یک فایل با متن ABRACADABRA دارید (رنگ‌آمیزی حروف صرفا برای تسهیل در خواندن است).

الف) به کمک روش هافمن، محتوای این فایل را به صورت باینری کدگذاری کنید. مراحل انجام کدگذاری را بنویسید. نیازی به نوشتن شبه کد الگوریتم هافمن نیست.

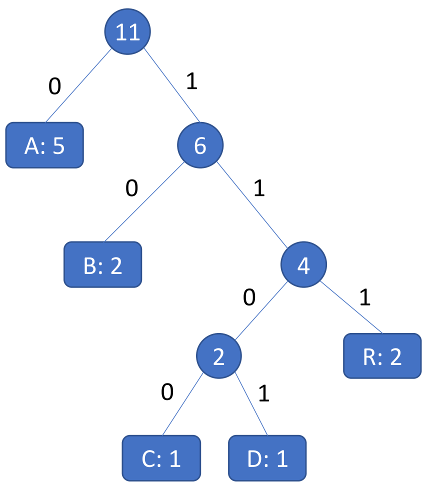
ب) میزان فشرده‌سازی (نسبت اندازه‌ی فایل فشرده شده به فایل اصلی) به کمک روش هافمن در مقایسه با ذخیره‌سازی به صورت متن چقدر است؟

ج) بهترین و بدترین میزان فشرده‌سازی برای هر رشته به کمک کدگذاری بدست آمده چقدر است؟ توضیح دهید.

نکته: هر بایت، ۸ بیت است.

پاسخ:

الف) (۱۰ نمره: ۶ نمره کد درست و درخت، ۴ نمره کدگذاری رشته) یکی از پاسخ‌های صحیح به کمک درخت زیر بدست می‌آید. پاسخ‌های درست دیگر با جابه‌جایی حروف با فراوانی مشابه (نظیر B و R) بدست می‌آیند.



حرف	A	B	C	D	R
کد	۰	۱۰	۱۱۰۰	۱۱۰۱	۱۱۱

بر این اساس، ABRACADABRA به صورت مقابل کد می‌شود:

۰۱۰۱۱۱۰۱۱۰۰۰۱۱۰۱۰۱۰۱۱۱۰

ب) (۴ نمره) دقت کنید که پاسخ این قسمت مستقل از جایگشت‌های جواب صحیح در قسمت (الف) است.

$$\frac{۲۳}{۱۱ \times ۸} = \frac{۲۳}{۸۸} \cong ۰.۲۶$$

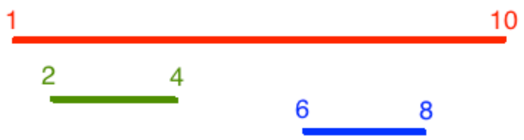
ج) (۶ نمره: ۳ نمره برای بهترین و ۳ نمره برای بدترین) بهترین فشرده سازی برای رشته‌هایی شامل حرف A اتفاق می‌افتد. در این رشته‌ها میزان فشرده‌سازی، $\frac{۱}{۸}$ است. بدترین فشرده‌سازی برای رشته‌هایی شامل C یا D اتفاق می‌افتد. میزان فشرده‌سازی این رشته‌ها، $\frac{۱}{۲}$ است.

---

۲. (۳۰ نمره) فرض کنید n بازه به صورت $I_i = [s_i, f_i]$ روی محور اعداد حقیقی داده شده‌اند. می‌خواهیم k نقطه به صورت $p_۱$، $p_۲$، ... ، p_k پیدا کنیم به طوری که به ازای هر i یک j وجود داشته باشد که $p_j \in I_i$. الگوریتم حریصانه‌ای پیشنهاد بدهید که مسئله را با کمترین k حل کند و شبه کد آن‌را بنویسید. بهینگی الگوریتم خود را ثابت کرده و پیچدگی زمان اجرای آن را بدست آورید.

پاسخ: ۱۰ نمره الگوریتم، ۷ نمره شبه کد، ۳ نمره تحلیل زمان اجرا، ۱۰ نمره اثبات

پاسخ حریصانه اشتباه: بازه‌ها را بر اساس زمان شروعشان مرتب می‌کنیم. سپس آن‌ها را به ترتیب بررسی می‌کنیم. اگر هر بازه با یک نقطه پوشانده نشده بود، نقطه شروع بازه را به لیست نقاط اضافه می‌کنیم. در شکل زیر، این راه ۳ نقطه ۱، ۲ و ۶ را انتخاب می‌کند در حالی که این مسئله با دو نقطه ۴ و ۸ قابل حل است.



پاسخ حریصانه صحیح: بازه‌ها را بر اساس زمان پایان‌یافتن مقایسه می‌کنیم. سپس به ازای هربازه اگر توسط نقطه‌ای پوشانده نشده بود، نقطه‌ی آخر بازه را به لیست نقاط اضافه می‌کنیم.

```
CoverIntervals(I) {
    I_sorted = sort_by_finish_times(I)
    points = []
    for each (s, f) in I_sorted {
        if points.size = 0 or points[-1] < s {
            points += [f]
        }
    }
    return points
}
```

زمان اجرای الگوریتم $O(n \log n)$ می‌باشد.

اثبات بهینه بودن: فرض کنید راه حل حریصانه (S_{greedy}) به پاسخ بهینه نرسد. بدون کاسته شدن از کلیت مسئله، فرض می‌کنیم نقطه‌ها در هر راه حل به صورت صعودی مرتب شده‌اند. شبیه‌ترین راه بهینه به راه حل حریصانه (راهی که بیشترین تعداد نقطه مشابه از ابتدا را با راه حریصانه دارد) در نظر بگیرید و آن‌را S_{opt} بنامید. فرض کنید اولین نقطه‌ای که این دو راه حل با هم تفاوت دارند، در نقطه kام اتفاق می‌افتد:

S_{greedy}: $x_1, x_2, \ldots, x_k, \ldots$

S_{opt} : $y_1, y_2, \ldots, y_k, \ldots$

با توجه به نحوه طراحی الگوریتم حریصانه، می‌دانیم که $y_k < x_k$ زیرا در الگوریتم حریصانه آخرین نقطه ممکن برای پوشش بازه مورد نظر انتخاب می‌شود. اگر y_k را از S_{opt} حذف کرده و x_k را به آن اضافه کنیم، باز هم همان پوشش بازه‌های را خواهیم

داشت. پس راه حل جدید بدست آمده هنوز بهینه است ولی به راه حریصانه شبیه‌تر شده است که این تناقض است. پس راه حل حریصانه، پاسخ بهینه را بدست می‌آورد.

---

۳. (۲۰ نمره) یک گراف وزن‌دار و درخت پوشای کمینه آن داده شده است. الگوریتم بهینه‌ای (به همراه شبه کد) برای هریک از حالات زیر ارائه دهید تا پس از تغییر گراف، درخت پوشای کمینه را به روزرسانی کند (به عبارتی درخت پوشا، کمینه باقی بماند.) تحلیل پیچیدگی زمان اجرای الگوریتم را نیز برای هر حالت بدست آورید.

الف) وزن یک یال گراف که جزئی از درخت پوشا کمینه داده شده نیست، کاهش بیابد.

ب) وزن یک یال گراف که جزئی از درخت پوشا کمینه داده شده است، کاهش بیابد.

پاسخ:

الف) (۱۵ نمره: ۸ نمره الگوریتم، ۴ نمره شبه کد، ۳ نمره زمان اجرا) یال با وزن کاهش یافته را به MST اضافه کنید. به کمک یک الگوریتم پیدا کردن دور (نظیر DFS)، دور ایجاد شده به واسطه یال جدید را پیدا کنید. یال با کمترین وزن را حذف کرده تا MST گراف جدید را بیابید.

```
UpdateMST(MST, w, e) {
    MST_new = MST + e
    cycle = Find_Cycle(MST_new)

    max_weight = -∞
    max_edge = null
    for each edge in cycle {
        if max_weight < w(edge) {
            max_weight = w(edge)
            max_edge = edge
        }
    }

    return MST_new - max_edge
}
```

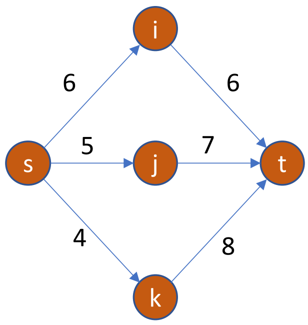
زمان اجرای الگوریتم، برابر $O(V+E)$ می‌باشد.

ب) (۵ نمره) چون MST شامل یال‌هایی با کمترین وزن است، کاهش وزن یال‌های آن باعث تغییر MST نمی‌شود.

---

۴. (۳۰ نمره) گراف جهت‌دار G با وزن‌های مثبت داده شده است. می‌خواهیم از بین کوتاه‌ترین مسیرهای از راس s به راس t، مسیری را پیدا کنیم که بیشترین وزن یال‌ها در این مسیر کمینه باشد. الگوریتم دایکسترا را طوری تغییر بدهید (به همراه ارائه شبه کد) که این مسیر را پیدا کند. پیچیدگی زمان اجرای الگوریتم خود را بدست آورده و نشان دهید پاسخ صحیح را پیدا می‌کند.

مثال: در گراف زیر، کوتاه‌ترین مسیر مورد نظر، $s \to i \to t$ می‌باشد. مسیرهای جایگزین، یعنی $s \to k \to t$ و $s \to j \to t$، هرکدام دارای یالی هستند که از بیشترین وزن یال‌های مسیر مورد نظر، وزن بیشتری دارد.



پاسخ: (۳۰ نمره: ۱۵ نمره الگوریتم، ۱۰ نمره شبه کد، ۵ نمره زمان اجرا)

مشابه مثال حل شده در کلاس، می‌توان متغیری به نام `max_edge[v]` تعریف کرد که طول بلندترین یال را در کوتاه‌ترین مسیر پیدا شده از s به v نگه دارد. هرگاه مسیری با طول مشابه و با یال کوتاه‌تر پیدا شد، `max_edge[v]` آپدیت شده و مسیر جدید به عنوان مسیر مورد نظر نگهداری شود. زمان اجرای این الگوریتم مشابه الگوریتم دایکسترا $O(|E|\log|V|)$ (در پیاده‌سازی زیر) یا $O(|V|^2)$ می‌باشد. برای چاپ جواب می‌توانید از مقدار ذخیره شده در متغیر `parent` استفاده کنید.

```
Dijkstra(G, w, s) {
    S = {}
    for each node v ∈ V
        dist[v] = ∞
        max_edge[v] = ∞  // new variable init
        parent[v] = null
    dist[s] = 0
    max_edge[s] = 0

    for each node v ∈ V
        Add v to Q with priority dist[v] **

    while Q ≠ {} {
        u = ExtractMin(Q) **
        S = S + {u}
        for each neighbor of u in V - S {
            // Updated condition
            if dist[v] > dist[u] + w(u, v) OR
             (dist[v] == dist[u] + w(u, v) AND max_edge[v] > max(max_edge[u], w(u,v))) {
                dist[v] = dist[u] + w(u, v)
                parent[v] = u
                // New variable update
                max_edge[v] = max(max_edge[u], w(u,v))
                Decrease priority of v in Q with dist[v]
            }
        }
    }
}
```