

مستند مراحل کلی توسعه نرم افزار دیکسترا

محمدیاسین داوده

۱۹ آبان ۱۳۹۸ | 11-10-2019

چکیده

The code is attached to the end of the document. **Notice that this code is one of my high school projects. Therefore, the quality of it is REALLY bad. Honestly, even I cannot take anything out of it. I just dumped it here with its original README (written in poor English).** Good luck with it!

این سند شرحی کلی بر مراحل توسعه پروژه سال دوم هنرستانم (۹۶-۹۷) — پیدا کردن بهینه ترین مسیر در شبکه — در رشته نرم افزار و شبکه است.

از آن جهت که به عنوان یک دانش آموز هنرستانی پیش از آن هرگز با چنین مسئله ای روبرو نشده بودم؛ می دانستم که هر اشتباه در مسیر توسعه می تواند به هزینه هنگفتی در زمان منجر شود. از همین رو مطالب مهم را در سندی تألیف کردم.

نسخه پیش رو ویرایشی از سند اصلی است که در \LaTeX باز نویسی شده است. این نسخه همچنین شامل بخش توضیحات اولیه و مفاهیم نیز می باشد.

توجه کنید که این کد و توضیحات قدیمی می باشند و کیفیتی مدرسه ای دارند! این سند صرفاً باز نویسی یادداشت های اصلی است. کد در انتهای سند است. (پ.ن: کیفیت کد به گونه ای است که من به عنوان نویسنده اصلی هم نمی توانم از آن استفاده کنم.)

فهرست مطالب

۲	۱ مفاهیم
۲	۲ اطلاعات اولیه
۲	۳ یادداشت های اولیه
۲	۱.۳ یادداشت اولیه
۳	۲.۳ یادداشت ثانویه
۳	۱.۲.۳ توضیحات
۶	۴ روش های حل مسئله
۶	۱.۴ روش اولیه (منطقی یا انسانی)
۱۰	۱.۱.۴ یادداشت روش منطقی
۱۰	۲.۱.۴ تجربه با روش انسانی
۱۱	۲.۴ روش ثانوی (نیاز به بازطراحی)
۱۱	۵ دیکسترا
۱۶	۶ Code

۱ مفاهیم

قبل از درک نرم افزار نیاز است مفاهیم پایه گراف ها را به خوبی بیاموزید. مطالب زیر برداشت هایی از Peter Linz در An introduction to Formal Languages and Automata است.

هر **گراف** ساختاری است که از دو مجموعه منتهی نقاط (Vertices) یا گره ها (Nodes) و یال ها (Edges) تشکیل می شود.

برای نمایش این اتصالات از **دیاگرام** های متفاوتی استفاده می شود؛ این دیاگرام ها غالباً از دایره برای نمایش نقاط، خطوط یا پیکان ها برای نمایش یال های جهت دار و بی جهت استفاده می کنند و مقیاس ندارند. مقیاس نداشتن به این معنا است که طول هر خط ارتباطی با طول یا وزن حقیقی اش ندارد و فقط نمایانگر اتصال است نه طول خط.

هر **یال (Edge)** از دو نود تشکیل می شود. به طور مثال: $e_i = (v_j, v_k)$ یالی است که از v_j به سمت v_k می رود به این معنا که جهتی یک طرفه به سمت v_j دارد به چنین یالی **کمان (Arc)** نیز می گویند. اگر که در تعریف هر گراف آمده باشد که گراف جهت دار یا هدایت شده (Directed) است، نمی توان دیگر در این دایگراف (Digraph) از v_k به v_j برگشت؛ مگر اینکه گراف جهت دار نباشد یا گراف از نوع مخلوط باشد — نوعی که شامل یال های دوطرفه و یک طرفه است — و یال مذکور دوطرفه تعریف شده باشد.

طول در گراف معنایی ندارد. مگر اینکه گراف از نوع موزون باشد.

یک **شبکه یا گراف موزون (Weighted)** گرافی است که در آن یال ها می توانند وزن یا به بیان دیگر هزینه پیمایش داشته باشند. موزون بودن گراف باعث می شود که مفهوم «طول یال» معنا پیدا کند.

هر **پیمایش (Walk)** ترتیبی از یال ها به صورت $(v_i, v_j), (v_j, v_k), \dots, (v_m, v_n)$ از v_i تا v_n است. در شبکه ها طول این پیمایش به اندازه جمع هزینه های یال ها است. در گراف ها این مقدار به اندازه تعداد یال ها است چرا که در اثر عدم وجود وزن، هر یال هزینه ای واحد (۱) دارد.

پیمایشی که در آن از هیچ یالی به صورت تکراری گذر نشود یک **مسیر (Path)** نامیده می شود.

مسیر ساده (Simple) مسیری است که در آن گذر از هیچ یالی تکرار نشود. از آنجایی که نرم افزار نهایی از حالات محاسباتی متفاوت و پیچیده ای پشتیبانی نمی کند، در تمام طول این سند (به جز این بخش) «مسیر» خلاصه ای از عبارت «مسیر ساده» است.

پیمایشی از نود v_i به خودش بدون پیمایش تکراری یالی یک **چرخه (Cycle)** با پایه (Base) است.

اطلاعاتی که به هر بخش از گراف منتسب می شوند **برچسب (Label)** نامیده می شوند. مانند نام نودها یا وزن یال ها. در نهایت، یالی از یک نود به خودش $e_i = (v_i, v_i)$ یک **حلقه (Loop)** خوانده می شود.

۲ اطلاعات اولیه

در زبان C# برنامه ای بنویسید که بهینه ترین مسیر ساده را در یک شبکه بی جهت پیدا کند.

زمان شروع: ۲۰۱۸/۳/۲۰ (۱۳۹۶/۱۲/۲۹)

آخرین تغییر قبل از ارائه: ۲۰۱۸/۳/۳۱ (۱۳۹۷/۱/۱۱)

۳ یادداشتهای اولیه

۱.۳ یادداشت اولیه

۱. نودها را دریافت کن (مثال ۵ عددی: e d c b a)

• هر نود رشته ای اختصاصی دارد که نشان می دهد به چه نقاط دیگری وصل است ($a \leftarrow cd$)

• هر نود آرایه ای دارد که هزینه جابه جایی برای هر یال روی آن مشخص است.

۲. مبدأ و مقصد را دریافت کن.

۳. تمام احتمالات را محاسبه کن و:

- (آ) کم هزینه‌ترین مسیر را پیشنهاد بده.
- (ب) پر هزینه‌ترین مسیر را پیشنهاد بده.

تودو

□ روتینگ دستی: کاربر بتواند انتخاب کند که از چه ایستگاه‌ها/نودهایی بگذرد.

۲.۳ یادداشت ثانویه

۱. تعداد نودها را دریافت کن (۵ عدد)

• آرایه‌ای 2×2 به شکل جدولی از اطلاعات با سطر و ستون‌هایی به اندازه تعداد نودهای گرفته شده از مرحله قبل بساز. هدر افقی و عمودی این جدول نام نودها در الفبای انگلیسی باشد.

۲. جدول را از قطر قرینه کن. (این مرحله جهت سهولت در ورودی گرفتن است. چرا که یال‌های جدول دوطرفه نیستند.)

۳. هر نود رشته‌ای اختصاصی دارد که نشان می‌دهد به چه نقاط دیگری وصل است ($a \leftarrow cd$)

۴. هر نود آرایه‌ای دارد که هزینه جابه‌جایی برای هر یال روی آن مشخص است.

۵. مبدأ و مقصد را دریافت کن.

۶. تعداد تمام احتمالات را از طریق فرمول زیر به دست می‌آید: $|Edges| = \frac{nodes \times (nodes - 1)}{2}$

- (آ) کم هزینه‌ترین مسیر را پیشنهاد بده.
- (ب) پر هزینه‌ترین مسیر را پیشنهاد بده.

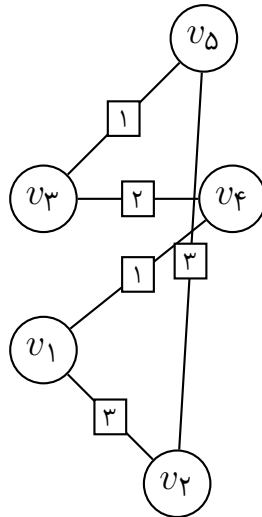
تودو

□ روتینگ دستی: کاربر بتواند انتخاب کند که از چه ایستگاه‌ها/نودهایی بگذرد.

□ با پیاده‌سازی اصولی و ورودی گرفتن باعث حذف مرحله ۲ شو و قابلیت جهت‌گیری را به برنامه اضافه کن.

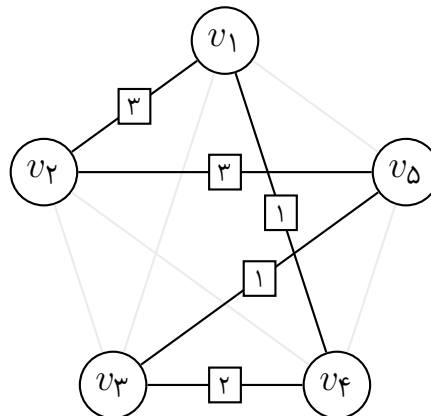
۱.۲.۳ توضیحات

از آنجایی که هر شبکه مجموعه‌ای از دو مجموعه است. ($Network = \{Vertices, Edges\}$) می‌توان آن را به صورت جدول نمایش داد.



شکل ۱: شبکه مفروض

اگر هر شبکه (مانند شکل ۱) را مرتب کنیم به یک چند ضلعی منتظم مشابه می‌شود و قوانین آن‌ها بر روی آن قابل اعمال است به عنوان مثال تعداد حداکثر یال را می‌توان از ضابطه زیر بدست آورد که فرمول محاسبه (اضلاع + قطرهای) به واسطه نقاط است. $|Edges| = \frac{nodes \times (nodes - 1)}{2}$



شکل ۲: شبکه مفروض مرتب شده

با توجه به موزون بودن شبکه تغییر شکل بصری دیاگرام در خصوصیات شبکه تأثیری نخواهد داشت. این ویژگی دیاگرام‌ها به ما این امکان را می‌دهد که هر دیاگرام را بتوانیم به صورت اصلی — همان مجموعه — به برنامه بدیم و بتوانیم هر شبکه را به صورت یک جدول (یا آرایه) هم به نمایش بگذاریم. در چنین جداولی هر سطر نماینده یک نقطه آغازین یک یال است و هر ستون نماینده یک نقطه پایانی برای همان یال است.

\times	v_1	v_2	v_3	v_4	v_5
v_1					
v_2					
v_3					
v_4					
v_5					

جدول ۱: نمونه جدول مشخصات یک شبکه مفروض

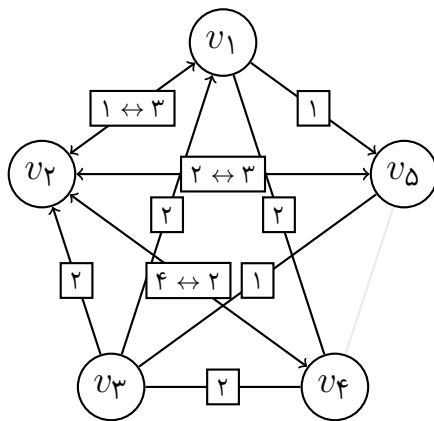
علت غیرفعال بودن قطر جداول آن است که پشتیبانی از حلقه‌ها در برنامه وجود ندارد. به همین دلیل نمی‌توان از یک نقطه به همان نقطه رفت.

برای ساختن هر یال جدید فقط کافیست که از اعداد حسابی در جدول استفاده کنیم. هر صفر نشان‌دهنده بسته بودن اتصال است و هر عدد طبیعی نشان‌دهنده وزن یال است.^۱

\times	v_1	v_2	v_3	v_4	v_5
v_1		۳	۰	۱	۰
v_2	۳		۰	۰	۳
v_3	۰	۰		۲	۱
v_4	۱	۰	۲		۰
v_5	۰	۳	۱	۰	

جدول ۲: اطلاعات پیاده شده شبکه‌های شکل‌های ۱ و ۲

نکته قابل توجه این است که به علت دو طرفه بودن شبکه‌ها اینگونه به نظر می‌رسد که هر جدول از قطر قرینه شده است.^۲



(ب) گراف مخلوط نمونه

\times	v_1	v_2	v_3	v_4	v_5
v_1		۱	۰	۲	۱
v_2	۳		۰	۲	۳
v_3	۲	۲		۲	۱
v_4	۲	۴	۲		۰
v_5	۰	۳	۱	۰	

(آ) جدول گراف مخلوط نمونه

شکل ۳: نمونه‌ای دیگر؛ اینبار از شبکه‌ای مخلوط

^۱ پ.ن: علت استفاده از اعداد حسابی آن است که در نسخه اصلی این سند، تا به اینجای کار از امکان وجود مسیرهای منفی در شبکه‌ها اطلاعی نداشته‌ام و در ادامه این کاوش، هنگامی که به انتخاب الگوریتم خواهیم رسید، بر حسب اتفاق الگوریتمی را انتخاب خواهیم کرد که از مسیرهای منفی پشتیبانی نمی‌کند. در نتیجه این تعاریف را دست نخورده رها می‌شوند.
^۲ در آینده از این ویژگی برای بهینه‌کردن فرآیند «دریافت ورودی» استفاده می‌کنیم. پشتیبانی از گراف‌های مخلوط هنوز به نسخه کنونی نرم‌افزار اضافه نشده است.

۴ روش‌های حل مسئله

برای حل مسائلی که کوتاه‌ترین مسیرهای ساده را در یک شبکه می‌خواهند باید اول درک کاملی از شبکه پیدا کنیم و پس از محاسبه تمام مسیرها بر اساس مبدأ مورد نظر به انتخاب مقصد پردازیم.

درک این موضوع فرآیند بسیار مهم است. پیش از ادامه از درک کامل این فرآیند اطمینان حاصل کنید. برای جلوگیری از ابهامات احتمالی به مثال زیر توجه کنید.

فرض کنید به خانه‌ای جدید نقل مکان کرده‌اید. منطق حکم می‌کند که برای روز مبدا نقشه فواصل محله جدید خود را به خاطر بسپارید تا اگر گاهی دیرتان شد بدانید بهترین مسیر کدام است (بهینه‌ترین مسیر). نقشه محله خود را دارید (اطلاعات گراف). آن را باز می‌کنید و به تمام نقاط بلوک‌تان مسیرها را محاسبه می‌کنید که از کدام کوچه‌ها به مکان‌های مهم سریع‌ترین دسترسی را دارید (محاسبه کوتاه‌ترین مسیر به تمامی نودها). پس از اینکه درکی کلی از کوتاه‌ترین مسیر به هر نقطه از بلوک یا محله‌تان را داشتید؛ برای پیمایش هر مسیر تنها کاری که نیاز است انجام دهید این است که آن مسیر را بپیمایید.

بدین ترتیب هرگاه که به خانه جدیدی نقل مکان کنید؛ یا بخواهید از مکانی دیگر از بلوک به خانه‌تان برگردید باید دوباره فواصل کل بلوک را به نسبت مبدأ دوباره محاسبه کنید.

تمام این مباحث برای نمایش دادن این است که برای محاسبه کوتاه‌ترین مسیر هر شبکه ابتدا لازم است که به نسبت مبدأ تمام مسیرها را محاسبه کرد. سپس انتخاب کرد که به کجا نیاز است بروید. اگر تمامی مسیرهای ممکن را محاسبه نکنید (حتی وقتی که این کار منطقی به نظر نمی‌رسد) ممکن است از وجود مسیری بهینه‌تر از بهینه‌ترین مسیر فعلی بی‌اطلاع بمانید.

مراحل کلی پیدا کردن بهینه‌ترین مسیر با منطق پیش-پردازش همه مسیرهای (درخت بهینه‌ترین مسیر) هر شبکه:

۱. محاسبه همه مسیرها به نسبت مبدأ

۲. تعیین مقصد

۳. فراخوانی مسیری که در مرحله ۱ محاسبه شده است.

۱.۴ روش اولیه (منطقی یا انسانی)

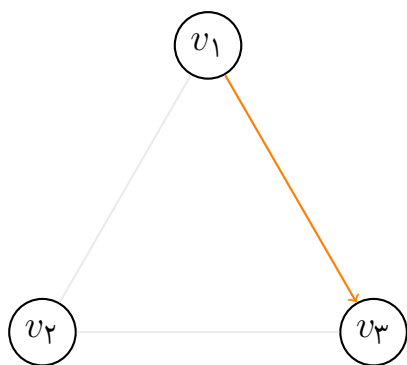
تنها روشی که اوایل ۲۰۱۸/۳ (۱۳۹۶/۱۲) به ذهنم رسید به این صورت بود که اولین کاری که باید برسانم دانستن تعداد تمام مسیرهای ممکن است. پس از آن محاسبه وزن هر مسیر و مقایسه آنهاست. از آنجا که این روش بر اصول یا اثبات ریاضی تأکیدی ندارد و صرفاً اولین راه حل منطقی‌ام برای حل مسئله بود؛ آنرا روش انسانی نامیده‌ام.

این توضیحات ارتباطی به راه حل نهایی ندارند می‌توانید به سادگی از آنها عبور کرد.

مثال‌ها و توضیحات زیر را مطالعه کنید. آیا می‌توانید الگوی رشد مسیرها را پیدا کنید؟ آیا منطق کاری را متوجه می‌شوید؟

اگر بدون در نظر گرفتن اتصالات بخواهیم صرفاً حداکثر مسیرهای هر شبکه را بدست آوریم (فرض مش) به طور مرتب شروع به شماردن تمام حالات می‌کنیم. برای جلوگیری از اشتباهات، با قاعده کردن و ماشینی کردن این فرآیند از منطقی خاص — پایه‌های متفاوت عددی — استفاده می‌کنیم که با مثالی در ادامه تشریح می‌شود.

از سه ضلعی شروع می‌کنیم. (شکل ۴)



(ب) گراف مخلوط نمونه

(آ) لیست مسیرها

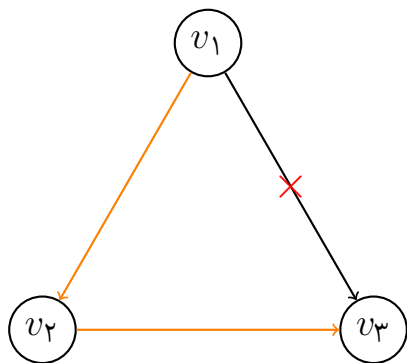
v_1, v_2	۱
------------	---

شکل ۴: شروع شمارش

در این مثال ابتدا مانند مبناهای عددی سعی می‌کنیم که کوتاه‌ترین مسیرها را بررسی کنیم. به این معنی که اول مسیرهای تک یالی را بررسی می‌کنیم؛ پس از آن مسیرهای دو یالی؛ پس از آن مسیرهای چند یالی و به همین صورت ادامه می‌دهیم تا همه مسیرها به دست آورده شوند.

اگر بخواهیم از v_1 به v_3 برویم؛ کوتاه‌ترین مسیر مستقیم دو نودی است: v_1, v_3 . در شکل ۴ این مسیر را لیست کرده‌ایم. حال این کار را انقدر تکرار می‌کنیم تا تمام مسیرهای دو نودی که از نود اول (v_1) شروع می‌شوند لیست شوند.

پس از محاسبه کوتاه‌ترین مسیرهای تک نودی از مبدأ به نود بعدی می‌رویم و از آنجا هر به پرداز تمام مسیرهای سه نودی ممکن می‌پردازیم. این فرآیند تا زمانی که همه نودها همه مسیرهای یکتای تک یالی را رفته باشند.



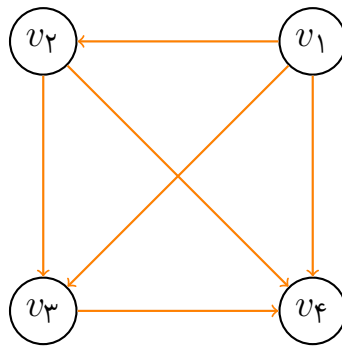
(ب) گراف مخلوط نمونه

(آ) لیست کامل مسیرها

v_1, v_2	۱
v_1, v_2, v_3	۲

شکل ۵: پایان شمارش

حال همین مثال را با گرافی چهار نودی تکرار می‌کنیم.



شکل ۶: گراف چهار نودی مخلوط نمونه

ابتدا مسیر دو نودی مستقیم به مقصد را می‌نویسیم: v_1, v_4
سپس مسیرهای سه نودی را بررسی می‌کنیم. (جدول ۳)

v_1, v_2, v_4	۱
v_1, v_3, v_4	۲

جدول ۳: لیست مسیرهای سه نودی ممکن از v_1 به v_4

اگر به الگوی بالا توجه کنید به این صورت است که به جز مبدأ و مقصد، عدد وسط ابتدا به حداکثر خود می‌رسد و سپس نود بعدی اضافه می‌شود. درست مانند اعداد. حال که نود بعدی اضافه می‌شود این فرآیند واضح‌تر می‌شود.

v_1, v_2, v_3, v_4	۱
v_1, v_3, v_2, v_4	۲

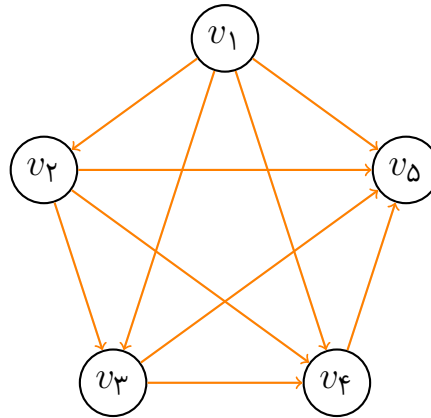
جدول ۴: لیست مسیرهای چهار نودی ممکن از v_1 به v_4

حال که مسیرهای چهار نودی نیز لیست شده‌اند حداکثر حالات ممکنه که می‌توان از v_1 به v_4 رسید به شرح زیر است.

v_1, v_4	۱
v_1, v_2, v_4	۲
v_1, v_3, v_4	۳
v_1, v_2, v_3, v_4	۴
v_1, v_3, v_2, v_4	۵

جدول ۵: لیست کامل مسیرهای چهار نودی

باز از سر، برای به دست آوردن حداکثر حالاتی که از v_1 به v_5 می‌توان به رفت؛ به همین صورت عمل کرد و ابتدا با مسیر دو نودی مستقیم از v_1 به v_5 و سپس با اضافه کردن نود و باز محاسبه، کار را به پایان رساند.



شکل ۷: گراف پنج نودی مخلوط نمونه

v_1, v_5	۱
v_1, v_2, v_5	۲
v_1, v_3, v_5	۳
v_1, v_4, v_5	۴
v_1, v_2, v_3, v_5	۵
v_1, v_2, v_4, v_5	۶
v_1, v_3, v_2, v_5	۷
v_1, v_3, v_4, v_5	۸
v_1, v_4, v_2, v_5	۹
v_1, v_4, v_3, v_5	۱۰
v_1, v_2, v_3, v_4, v_5	۱۱
v_1, v_2, v_4, v_3, v_5	۱۲
v_1, v_3, v_2, v_4, v_5	۱۳
v_1, v_3, v_4, v_2, v_5	۱۴
v_1, v_4, v_2, v_3, v_5	۱۵
v_1, v_4, v_3, v_2, v_5	۱۶

جدول ۶: لیست کامل مسیرهای پنج نودی ممکن از v_1 به v_5

بدین صورت و با این روش می‌توانیم تمام مسیرهای بین دو نقطه را در هر مش به دست آوریم. اگر به این روند افزایشی دقت کافی کنیم می‌توانیم آنرا به اعداد تشبیه کنیم. اعدادی که صفر ندارد و مبنایی به تعداد بعلاوه یک حداکثر نودهای شبکه دارد. ($Base = |Vertices| + 1$) به طور مثال اگر گراف مفروض ۵ نود داشته باشد. سامانه عددی را با مبنای شش در نظر می‌گیریم و با قوانین زیر شروع به عدد سازی با آن می‌کنیم.

۱. عددها از ۰ شروع نمی‌شوند و تا N نمی‌روند.

۲. عددها همیشه با عدد نود مقصد شروع می‌شود و به نود مبدأ خاتمه می‌یابد.

با توجه به نتیجه‌گیری‌هایی که در این بخش شد مشخص است که الگوریتم این روش باید بتواند مبناهای مختلف را ایجاد کند و آنرا بین مبدأ و مقصد قرار دهد و اگر عدد یا رشته‌ای از حروف تکراری تشکیل نشده بود به شمار مسیرهای مجاز رشته را اضافه کند.

از اینجا به بعد با مقایسه جمع هزینه‌های هر مسیر به این نتیجه می‌رسیم که کدام مسیر کوتاه‌تر است.

۱.۱.۴ یادداشت روش منطقی

در این یادداشت با راه حلی که داریم یادداشت‌های قبلی خود را قدمی بیشتر به الگوریتمی قابل پیاده‌سازی تبدیل می‌کنیم.

۱. تعداد نودها را دریافت کن. (مثال: ۵ عدد)

۲. جدولی از اطلاعات گراف بساز.

۳. اطلاعات جدول را ورودی بگیر.

۴. از قرینه بودن جدول از قطر اطمینان حاصل کن.

۵. مبدأ و مقصد را دریافت کن.

۶. تمام مسیرهای ممکن را محاسبه کن: رشته‌ای به شکل

$[d][Variant][s]$

بساز که در آن $[d]$ مقصد است. بخش $[Variant]$ بخش متغیر مسیر است و $[s]$ نود مبدأ است.

۷. لیستی خالی بساز که مسیرها در آن قرار گیرند.

۸. مسیر مستقیم مبدأ تا مقصد را به لیست اضافه کن.

۹. با توجه به قالب تعریف شده در مرحله ۶ از ۱ تا $|Vertices|$ شروع به شمارش کن و به ازای هر عدد:

(آ) اگر عدد از تعداد نودها کوچکتر بود و در رشته وجود نداشت (تکراری نبود):

i. اگر تعداد حروف رشته کوچکتر از تعداد نودها بود:

آ. رشته را در لیست ثبت کن.

ب. به مرحله ۹ بازگرد.

ii. در غیر این صورت به مرحله بعد برو.

(ب) جمع هزینه هر مسیر لیست را به دست بیاور.

(ج) با مقایسه هزینه‌ها کوتاه‌ترین مسیر را معرفی کن.

۲.۱.۴ تجربه با روش انسانی

با نوشتن برنامه این روش متوجه می‌شوید که این روش مشکلاتی دارد.

به طور تخمینی تا ۱۰ نود را به خوبی شناسایی می‌کند و خروجی مناسب را با توجه به تمام حالات ممکن می‌دهد.

اما با توجه به رشد تساعدی تعداد مسیرهای ممکن برای پیمایش به سوی مقصد، پس از ۱۰ نود، تعداد حالات به حدی زیاد می‌شوند که اگر برنامه سرریز (Overflow) نکند؛ پردازش آن تا ۳۰ دقیقه نیز ممکن است طول بکشد.

این دلیلی بود تا در تاریخ ۲۰۱۸/۳/۱۵ (۱۳۹۶/۱۲/۲۴) به این نتیجه رسیدیم که بهینه کردن این الگوریتم یا روش بی‌تأثیر است چرا که از پایه روشی غلط برای حل این مسئله است و با بازنگری‌هایی کودکانه و با حداقل دانش ریاضی نمی‌توان الگوریتمی بهینه برای حل این مسئله نوشت.

در سند اصلی این بخش با خط زیر به پایان می‌رسد:

«متأسفانه این برنامه اولین برنامه‌ای بود که از ذهن خودم راه‌حلی را توسعه ندادم و مجبور به تحقیق میدانی و تخصصی درباره الگوریتم هستم. با کمک منابع خارجی و مطالعه آزاد الگوریتم ویرایش خواهد شد و ادامه گزارش نوشته خواهد شد.» و خوشبختانه تحقیقات و مطالعاتم نتیجه داد و در ادامه این الگوریتم ویرایش شد و ادامه گزارش در سند اصلی نوشته شد که به شرح زیر است:

۲.۴ روش ثانوی (نیاز به بازطراحی)

پس از شکست روش اولیه بنا بر این که برنامه را با یک الگوریتم جدید از صفر دوباره طراحی کنم و اینبار تنها به دانش ناقص خودم تکیه نخواهم کرد و از منابع آزاد برداشت‌هایی خواهم کرد. پس از بررسی‌هایی سطی درباره الگوریتم‌های حلال مسئله‌های «کوتاه‌ترین مسیرها تک-منبعی/آغازی» (Single-Paths) Shortest Source به این نتیجه می‌رسیم که سابقاً الگوریتم‌هایی در رابطه با چنین مسائلی توسعه داده شده است. از این دسته الگوریتم‌ها می‌توان به الگوریتم‌های زیر اشاره کرد:

• الگوریتم Bellman-Ford

- با تأکید بر روزرسانی و پیمایش چندین و چند باره تمام شبکه
- برای پیدا کردن کوتاه‌ترین مسیر در شبکه‌هایی که شامل چرخه‌های منفی نیست.

• الگوریتم Dijkstra

- با تأکید بر افزایش بهینگی و دقت در مقایسه با الگوریتم Bellman-Ford
- برای پیدا کردن کوتاه‌ترین مسیر در شبکه‌هایی که شامل یال‌های منفی نیست.
- این الگوریتم «درخت کوتاه‌ترین مسیر» را روی هر گراف طراحی می‌کند. در نتیجه با فراخوانی یادداشت‌های هر نقطه تا مبدأ می‌توانیم به کوتاه‌ترین مسیر تا آن نقطه دست پیدا کنیم. و این مطلب فقط برای مقصد مشخص نیست. بلکه از مبدأ مشخص همه نقاط به دست می‌آیند. برای جلوگیری از پردازش اضافه با طراحی شیب (مطابق الگوریتم A^*) می‌توانیم از پردازش اضافه جلوگیری کنیم.

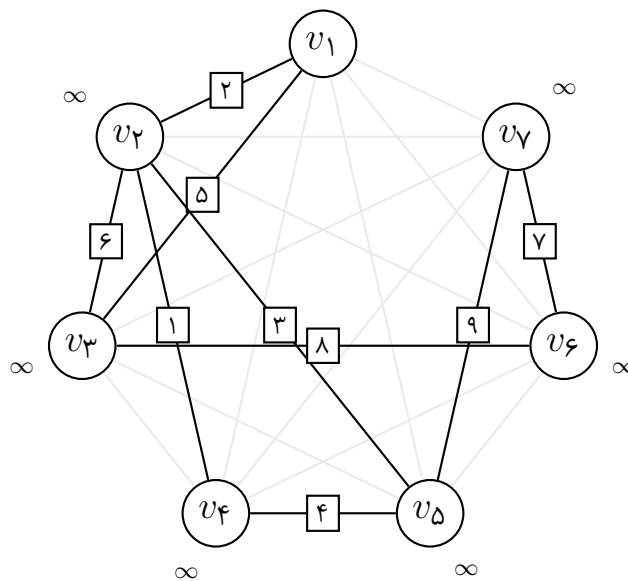
• الگوریتم A^* (A-Star)

- با تأکید بر افزایش دقت در مقایسه با الگوریتم Dijkstra
 - برای پیدا کردن بهینه‌ترین مسیر در شبکه‌ها با استفاده از تکنیک‌های شبیه‌سازی شیب و جهت جغرافیایی
- پس از بررسی‌هایی در منابع غیرفارسی الگوریتم Dijkstra (دیکسترا) که ترکیبی بی‌نقص از بهینگی، دقت و پیچیدگی را داشت را انتخاب کردم.^۴

۵ دیکسترا

همانطور که پیشتر مختصراً اشاره شد؛ الگوریتم دیکسترا (Dijkstra) به کاربران کمک می‌کند تا در شبکه‌هایی که در آن‌ها یال‌هایی منفی وجود ندارد کوتاه‌ترین مسیر را به دست بیاورند.

^۴ علاوه بر سادگی در اجرا دلایلی شخصی برای انتخاب الگوریتم دیکسترا در مقایسه با دیگر انتخاب‌ها بود که در حوزه بحث این سند نمی‌گنجد.



شکل ۸: گراف نمونه

برای این الگوریتم نیز می‌توانیم از مثال‌های جغرافیایی استفاده کنیم: هر نود شهری است که با راهی به شهر دیگر متصل است. می‌خواهیم از کوتاه‌ترین مسیر از شهر v_1 به شهر v_8 بریم (شکل ۸).

ابتدا، لیستی از تمام نودها به همراه سه برچسب برای هرکدام تهیه می‌کنیم. این سه برچسب از این قرار است:

۱. کوتاه‌ترین مسیر از نقطه شروع تا آن نقطه چه مقدار است؟

۲. در کوتاه‌ترین مسیر، چه نقطه‌ای قبل از (پدر) این نقطه بوده است؟

۳. تابلحال این این نقطه بررسی شده است؟

در فرض اولیه برای شروع کار الگوریتم:

۱. بیشینه کوتاه‌ترین مسیر را به عنوان فاصله هر نقطه در نظر می‌گیریم. این بیشینه معمولاً بی‌نهایت (∞) در نظر گرفته می‌شود.

۲. — از آنجایی که هیچ مسیری را بررسی نکرده‌ایم — نقطه قبل از (پدر) هر نقطه را خالی در نظر می‌گیریم.

۳. — از آنجایی که هیچ نقطه‌ای را بررسی نکرده‌ایم — وضعیت همه نقاط را «بازدید نشده» در نظر می‌گیریم.

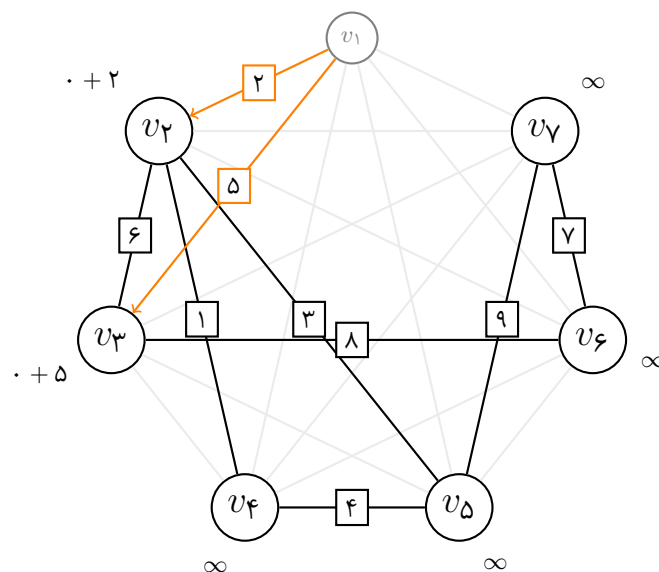
با داشتن مفروضات اولیه الگوریتم می‌تواند شروع به کار کند.

فاصله تمام یال‌هایی که از مبدأ قابل دسترسی هستند را می‌گیریم. (مثال: v_1, v_2). به ازای هر نود که اندازه گرفتیم:

۱. اگر طول مسیری که از پدر به آن نود می‌رسد کمتر از برچسب مسیر سابق بود آنرا به روز رسانی می‌کنیم. با بروزرسانی این برچسب، برچسب «نود پدر» را هم بروزرسانی می‌کنیم.

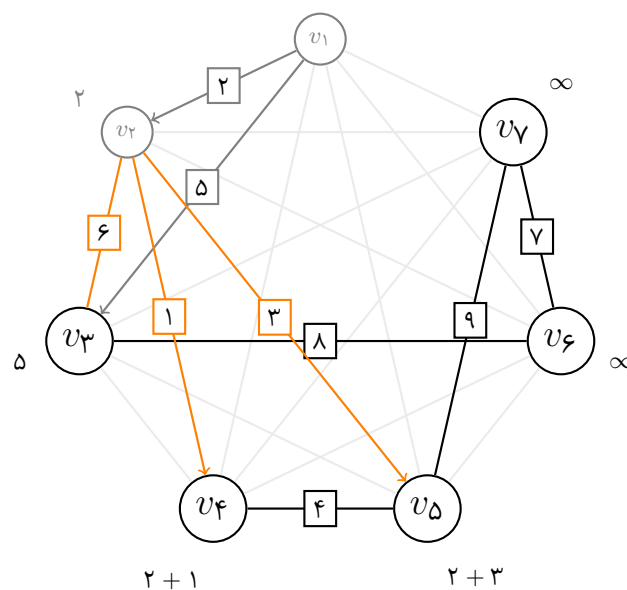
۲. اگر تمام نودهای متصل به مبدأ بررسی شدند. نود مبدأ را «بررسی شده» در نظر می‌گیریم.

و این فرآیند بارها تکرار می‌شود. هر بار نودی که کمتر مسیر را دارد و وضعیت آن «بررسی نشده» است اندازه‌گیری می‌شود تا تمام گراف به نسبت مبدأ پردازش شود.



شکل ۹: گراف نمونه که در آن دو نود بررسی شده‌اند و نودی که همه یال‌های متصل به آن بررسی شده‌اند در حالت «بررسی شده» قرار گرفته است.

برای این مثال نودی که هنوز بررسی نشده و کمترین هزینه را دارد - در این مورد v_2 - انتخاب می‌کنیم.



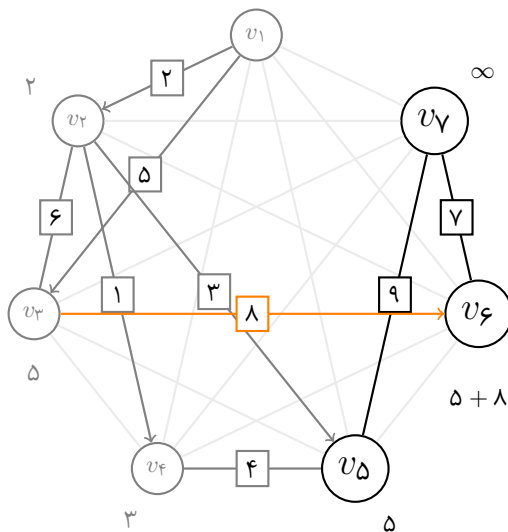
شکل ۱۰: گراف نمونه که در ادامه بررسی، از نودی که کمترین هزینه را داشته است (v_2 بین v_2 و v_3) فرآیند بررسی و برجسب‌گذاری را دوباره انجام می‌دهیم.

پس از انتخاب نود بعدی از بین نودهای متصل به نود فعلی (v_2 از بین نودهای متصل به v_1 یعنی v_1, v_2) شروع به تکرار فرآیند پیشتر ذکر شده می‌کنیم که در زیر دوباره تشریح شده است. (مطابق شکل ۱۰)

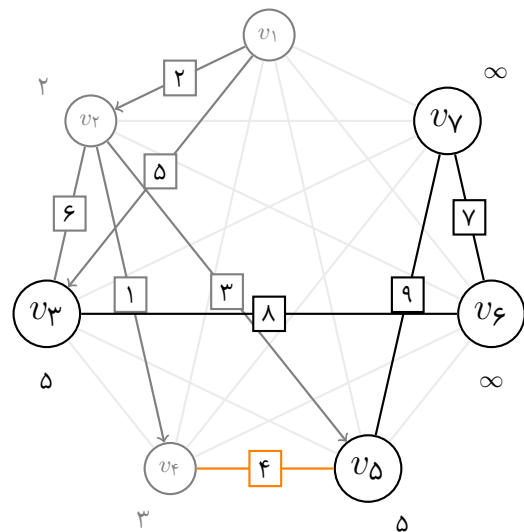
۱. بررسی می‌کنیم که از v_2 به چه نودهایی دسترسی داریم؟ (v_3, v_4, v_5)

۲. به ترتیب برای هر نودی که به آن دسترسی داریم:

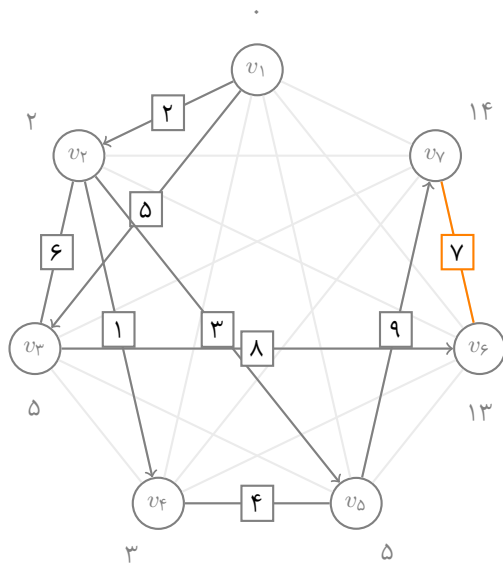
- (آ) اگر طول مسیری که از پدر به آن نود می‌رسد کمتر از برچسب مسیر سابق بود آنرا به روز رسانی می‌کنیم. با بروزرسانی این برچسب، برچسب «نود پدر» را هم بروزرسانی می‌کنیم.
 (در اولین مثال: آیا $2 + 6$ از هزینه قبلی نود (۵) کمتر است؟ چون جواب خیر است کاری به نود نداریم).
 (در دومین مثال: آیا $2 + 1$ از هزینه قبلی نود (۵) کمتر است؟ چون جواب بله است برچسب قدیمی (∞) حذف می‌شود و هزینه جدید را ثبت می‌کنیم. همچنین یادداشت می‌کنیم که پدر v_4 نود v_2 است).
 (ب) اگر تمام نودهای متصل به مبدأ بررسی شدند. نود مبدأ را «بررسی شده» در نظر می‌گیریم.
 تمام این پروسه را آنقدر تکرار می‌کنیم تا همه نودها به حالت «بررسی شده» درآیند.



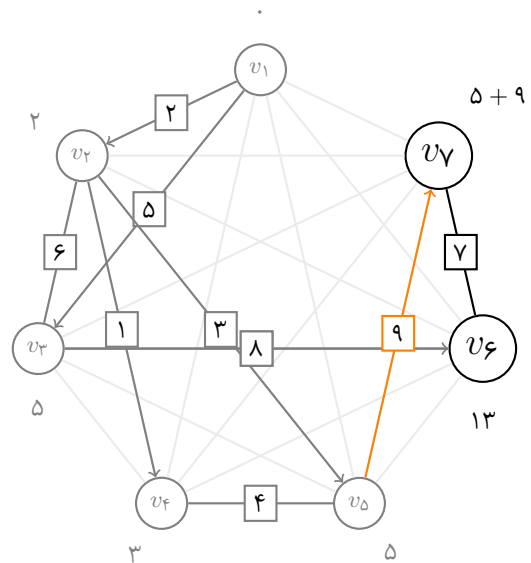
شکل ۱۲: گراف نمونه در ادامه بررسی



شکل ۱۱: گراف نمونه در ادامه بررسی

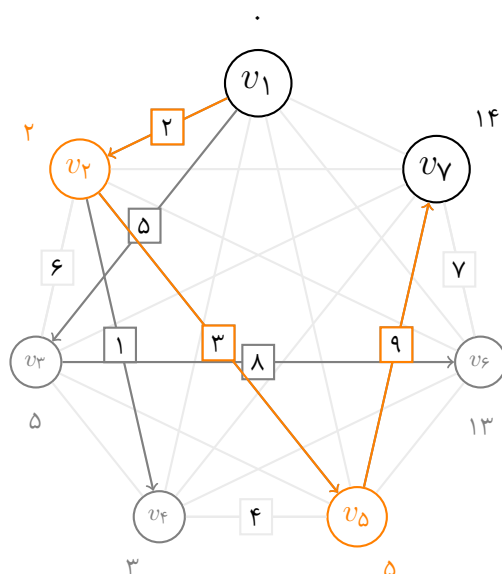


شکل ۱۴: گراف نمونه در آخرین مرحله بررسی



شکل ۱۳: گراف نمونه در ادامه بررسی

هنگامی که همه نودها بررسی شدند با بررسی پدر مقصد انتخابی به صورت بازگشتی (Recursive) به نود مبدأ می‌رسیم. مسیری که پیموده می‌شود کوتاه‌ترین مسیر است. اگر مبدأ در این فرآیند بررسی پدر پیدا نشد یعنی از نود مبدأ دسترسی به مقصد غیرممکن است. (مثال شکل ۱۵)



شکل ۱۵: با دنبال کردن پدر نود نهایی به صورت بازگشتی بهترین مسیر مشخص می‌شود.

```

1 // By M. Yas. Davoodeh
2 // This is my first program written in C# as a highschool project.
3 // THE QUALITY OF IT IS NOT ACCEPTABLE! NOTICE AGAIN: THIS IS A HIGH SCHOOL LEVEL CODE!
4 /*
5  Some notes:
6
7  Handwritten/Hardcore Codes: codes which are in noobish style and not flexible.
8  They are so easy to break. Check every single one of them if you make any change to the
   ↪ program.
9
10 Modes: This little program supports two modes of inputs. One (Default) helps you
11 with creating a graph in peace and breaks relatively hardly (for users). The Second one
12 helps you to develop and input a graph ASAP.
13 In the latter mode you have to make a flawless "String" made specifically for this mode
14 and use it as input. I made this mode for me and it is super buggy!
15 To make a "Input string" you basically use every input separated with a space, in order.
16 If you provide the program with such a string a Bot function will Enter
17 String Mode Syntax:
18 `[NodesCount] [theCostOfEdge1] [From] [To] [theCostofEdge2] [From] [To] ...
19 [-3 (just when you are done with entering inputs)] [StartNode] [DestinationNode]`
20 e.g. The First Graph in my Documentation about Dijkstra:
21 `7 2 a b 5 a c 6 b c 1 b d 3 b e 4 d e 8 c f 9 e g 7 f g -3 a g`
22
23 Extra: Extra things are just Extra (like Updates) and *I* even don't need them
24 in anycase BUT I thought it's better to keep them here for further development.
25 (because back then I didn't know such thing as Git exists.)
26
27 This program only supports inputs up to 20 nodes.
28 */
29
30 using System;
31 using System.Diagnostics.CodeAnalysis;
32 using System.Text.RegularExpressions;
33
34 namespace router
35 {
36     internal class Program
37     {
38         private const bool Error = true, // When true user receives
           ↪ "Error"s and Feedbacks
39
40         Reportavailable = true; // When true you can call
           ↪ table by -1 in most inputs

```



```

F0 private const byte Lettercode = 64, // 64 for CAPs 96 for
    ↪ small-caps letter signs
F1 Maxnodetosupport= 26; // as letters are 26
    ↪ //TODO extend me
F2 private const int Infinite = int.MaxValue; // Extend Infinite when
    ↪ extend var type : Handwrite*
F3 public enum Alerts { Error, Info, Notice, Success, Default, DefaultWithBR }
F4 private static Dijkstra.Node[] _dijkstraNodes;
F5 private static int[,] _nodesTableInts; // The Graph states and
    ↪ Definition is goes in this table
F6 private static int _stringNthRequired = 1; // is used in StringMode
    ↪ and makes string into Fragments
F7 private static byte _nodesCount; // Number of Nodes
F8 private static string _string;
F9 private static char _sourceNode,
D0 _destinationNode;
D1 private static bool _stringMode;
D2
D3 private static bool Mapper ()
D4 {
D5 //Mapper mode to Input the network you want to find shortest path in it
D6 ↪ (Mapper Mode)
D7
D8 //Intro
D9 string header_temp = "Mapper Mode";
DA Output.Visuals.HeaderFooter.Start(header_temp);
DB
DC //Input: Number of Nodes
DD string output_temp = "Enter the number of nodes on your network: ";
DE if (!_stringMode)
DF _nodesCount = (byte)Input.Int32(output_temp, 3, Maxnodetosupport,
    ↪ checkReportPossible: false);
DA else
DB {
DC _string = Input.String(
DE "(If you don't know what you are doing run DefaultMode by entering 1)"
    ↪ + "\n" +
DF "Input THE string you want to apply:", Alerts.Notice, false);
DA _nodesCount = byte.Parse(Job.ExtractNumbers(Input.Bot(output_temp)));
DB }
DE
DE //Initialize Table of Network States or NodeTable
DA _nodesTableInts = new int[_nodesCount + 1, _nodesCount + 1]; //+1 because of
    ↪ Table header and labels

```

```

76 //fill the table/array with headers and labels
77 _nodesTableInts[0, 0] = -1;
78 for (int i = 1; i ≤ _nodesCount; i++)
79 {
80     _nodesTableInts[0, i] = i + Lettercode;
81     _nodesTableInts[i, 0] = i + Lettercode;
82     _nodesTableInts[i, i] = -1;
83 }
84
85 //Input: Data Entry (for NodeTable)
86 Output.Alert("START: Data entry", Alerts.Notice);
87 while (true)
88 {
89     output_temp = "Enter the Cost-Price you want (-3 to break): ";
90     int data =
91         _stringMode? int.Parse(Job.ExtractNumbers(Input.Bot(output_temp))) :
92         ↪ Input.Int32(output_temp, -3);
93
94     if (data < 0)
95     {
96         if (data == -3)
97         {
98             Output.Alert("END: Data entry", Alerts.Notice);
99             break;
100         }
101         continue;
102     }
103
104     char x, y;
105     if (_stringMode)
106     {
107         x = char.Parse(Input.Bot("From: "));
108         y = char.Parse(Input.Bot("To: "));
109     }
110     else
111     {
112         char endNodeChar = (char)(_nodesCount + Lettercode);
113         x = Input.Char("From: ", max: endNodeChar);
114         y = Input.Char("To: ", max: endNodeChar);
115     }
116     if (x ≠ y)
117     {
118         _nodesTableInts[x - Lettercode, y - Lettercode] = data;
119         _nodesTableInts[y - Lettercode, x - Lettercode] = data;

```

```

120         continue;
121     }
122     // ReSharper disable once ConditionIsAlwaysTrueOrFalse
123     if (Error) Output.Alert("You cannot move from a place to itself. That's
    ↳ non-sense.", Alerts.Error);
124     Output.BR();
125 }
126
127
128 //Outro
129 Output.Visuals.HeaderFooter.End(header_temp);
130 Output.BR();
131 if (_stringMode)
132     Output.Alert("There is Your Network! You are all done here.\n" +
133         "You'll go to next session now." + "\n" +
134         "Where you can process this network (print after this
    ↳ message) you made.", Alerts.Success);
135
136 else
137     Input.String("There is Your Network! You are all done here.\n" +
138         "Press anykey to go to next part." + "\n" +
139         "Where you can process this network (print after this
    ↳ message) you made.\n" +
140         "Press any key to Continue... ", Alerts.Success);
141
142 Output.BR();
143 return true;
144 }
145
146 private static bool Router()
147 {
148     //Router Mode to find the shortest map you gave before in Mapper Mode (Router
    ↳ Mode)
149     //Intro
150     string header_temp = "Router Mode";
151     Output.Visuals.HeaderFooter.Start(header_temp);
152
153     //Inputs: Start/Source Node and Destination Node
154     string output_temp = "Enter start Node: ";
155     if (_stringMode)
156     {
157         _sourceNode = char.Parse(Input.Bot( output_temp ));
158         _destinationNode = char.Parse(Input.Bot( "Enter destination node: "));
159         if (_sourceNode == _destinationNode)
160         {
161             Output.Alert(

```

```

160         "FATAL: SOURCE AND DESTINATION CANNOT BE THE SAME. (FATAL ONLY IN
        ↪ STRING MODE)", Alerts.Error);
161     return false;
162 }
163 }
164 else
165 {
166     char endNodeChar_temp = (char)(_nodesCount + Lettercode);
167     _sourceNode = Input.Char(output_temp, max: endNodeChar_temp);
168     while (true)
169     {
170         _destinationNode = Input.Char("Enter destination node: ", max:
        ↪ endNodeChar_temp);
171         if (_sourceNode == _destinationNode)
172             Output.Alert("Source and Destination cannot be the same.",
        ↪ Alerts.Error );
173         else
174             break;
175     }
176 }
177
178 //Check possibility
179 Output.Alert("Checking Connections...", Alerts.Info);
180 if (!Job.CheckNodeConnection(_sourceNode))
181 {
182     Output.Alert("FATAL: START/SOURCE NODE IS DISCONNECTED FROM THE NETWORK!"
        ↪ , Alerts.Error );
183     return false;
184 }
185 if(!Job.CheckNodeConnection(_destinationNode))
186 {
187     Output.Alert("FATAL: DESTINATION NODE IS DISCONNECTED FROM THE NETWORK!"
        ↪ , Alerts.Error );
188     return false;
189 }
190 Output.Alert("Start Node and Destination Node are Connected to the
        ↪ Network...", Alerts.Success);
191 Console.WriteLine("Let's see if it's possible to find Shortest Path from Start
        ↪ Node to Destination Node." );
192
193 //Dijkstra Algorithm
194 Output.Alert("START: Dijkstra's Process", Alerts.Notice);
195 Dijkstra.Run();
196
197 Output.Visuals.HeaderFooter.End(header_temp);

```

```

198         return true;
199     }
200     public static void Main()
201     {
202         //Intro
203         Output.Alert("Usually, when input available," + "\n" +
204             "\t" + "you can get a Table of Network States (NodeTable) by
205             ↪ entering \"-1\".", Alerts.Info );
206
207         //Input: Mode*
208         Output.Alert("How you wanna enter output?"
209             ↪ , Alerts.Notice);
210         Output.Alert("CAUTION: String Mode (2) Has No Debugger and is NOT stable..." +
211             ↪ "\n" +
212             "\t" + "It's generally for test and Develop..." +
213             "\t" + "Do NOT use it if you are not sure about inputs."
214             ↪ Alerts.Error, false );
215         int inputMode_temp =
216             Input.Int32("Enter Mapper Mode: (Default: '1' or String: '2') ", 1, 2,
217                 ↪ checkReportPossible: false);
218         switch (inputMode_temp)
219         {
220             case 1:
221                 _stringMode = false;
222                 break;
223             case 2:
224                 _stringMode = true;
225                 break;
226         }
227
228         //Mapper mode to Input the network you want to find shortest path in it
229         ↪ (Mapper Mode)
230         if (!Mapper())
231             return;
232
233         Output.Visuals.Table.Int(_nodesTableInts);
234
235         //Router Mode to find the shortest map you gave before in Mapper Mode (Router
236         ↪ Mode)
237         if (!Router())
238             return;
239
240         Console.ReadKey();
241     }

```

```

236
237     private class Dijkstra
238     {
239
240         public class Node
241         {
242             //public string Name; //For Update*
243             public char Status = 't', //Status can be 'p' (Permanent) or 't'
244                 ↪ (Temporary) TODO explain more
245             ForeNode;
246             public int PathCost;
247         }
248
249         //We use a -1 when it comes to _dijkstraNodes Array
250         //because array starts at 0 but first array (nodeTable) starts at 1
251
252         private static void Init()
253         {
254             //Initialize
255             //Dijkstra: at first we guess all Nodes' PathCost. Source Node = 0 and
256             ↪ rest are Infinite
257             _dijkstraNodes = new Node[_nodesCount];
258             for (int i = 0; i < _nodesCount; i++)
259             {
260                 _dijkstraNodes[i] = new Node
261                 {
262                     //Name = (i + Lettercode + 1).ToString(),
263                     ForeNode = '-',
264                     PathCost = Infinite
265                 };
266             }
267             _dijkstraNodes[_sourceNode - Lettercode - 1].Status = 'p';
268             _dijkstraNodes[_sourceNode - Lettercode - 1].PathCost = 0;
269         }
270
271         private static char[] ReturnNodesToSelectMinBetween(bool
272             ↪ FirstTimeFor_sourceNode)
273         {
274             string nodesToSelectMinBetween = "";
275             char[] tempNodesConnectedTo = Get.TempNodesConnectedTo(_sourceNode);
276             if (FirstTimeFor_sourceNode)
277                 nodesToSelectMinBetween = Get.StringFromArray
278                 ↪ (tempNodesConnectedTo);
279             else

```

```

277         for (int i = 0; i < _nodesCount; i++)
278         {
279             char status;
280             try { status = _dijkstraNodes[i].Status; }
281             catch { continue; }
282             if (status == 't')
283                 nodesToSelectMinBetween += (char) (i + Lettercode + 1);
284         }
285         return Get.CharArrayFromString(nodesToSelectMinBetween);
286     }
287
288     private static char SelectMinChar(string nodesToSelectMinBetween, int shift =
289     ↪ 0)
290     {
291         //Dijkstra: Between Candidate nodes (from first step) we select the one
292         ↪ with Lowest Cost (minChar = )
293         //We Turn the MinChar (Return of this Function) into next CurrentNode
294         //Shift doesn't let program to do any loops if all values are same.
295         //If there are two candidates for minChar first time it will select first
296         ↪ next time it will select next.
297
298         int c = 0;
299         int min = 0;
300         char minChar = '0';
301         foreach (char node in nodesToSelectMinBetween)
302         {
303             if (c == shift)
304             {
305                 min = _dijkstraNodes[node - Lettercode - 1].PathCost;
306                 minChar = node;
307             }
308             else if (c > shift)
309             {
310                 if (_dijkstraNodes[node - Lettercode - 1].PathCost < min)
311                 {
312                     min = _dijkstraNodes[node - Lettercode - 1].PathCost;
313                     minChar = node;
314                 }
315             }
316             c++;
317         }
318         return minChar;
319     }
320 }
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500

```

```

319 private static char[] FindForeNodes(char node, string path_memory = "")
320 {
321     //This Function Finds the Path from _destinationNode back to the
322     ↪ source/start point
323     char foreNode = _dijkstraNodes[node - Lettercode - 1].ForeNode;
324     if (foreNode != '-')
325     {
326         path_memory += foreNode;
327         return FindForeNodes(foreNode, path_memory);
328     }
329     char[] pathChars = new char[path_memory.Length + 1];
330     int counter = path_memory.Length - 1;
331     foreach (char station in path_memory)
332         pathChars[counter--] = station;
333
334     pathChars[path_memory.Length] = _destinationNode;
335     return pathChars;
336 }
337
338 public static void Run()
339 {
340     Init();
341     //Dijkstra: Start from _sourceNode (start point). we set it to
342     char currentNode = _sourceNode;
343     Output.Alert("currentNode = " + currentNode, Alerts.Info);
344     bool FirstTimeFor_sourceNode = true;
345     string error = "";
346     int shift = 0;
347     char minChar = '-';
348     //Dijkstra: Then we select first Node that is connected to _sourceNode
349     ↪ (start Point)
350     while (true)
351     {
352         if (error == "fragmental Network") break;
353         string nodesToSelectMinBetween =
354
355             ↪ Get.StringFromArray(ReturnNodesToSelectMinBetween(FirstTimeFor_sourceNode
356         if (nodesToSelectMinBetween != "")
357             Output.Alert(
358                 "Nodes To Select a Minimum Between them are: " +
359
360                 ↪ Output.Arrays.Char(Get.CharArrayFromString(nodesToSelectMinBetween),
361                 ↪ false)
362             , Alerts.Info);

```



```

359     else
360     {
361         Output.Alert
362             ("There is No Node Left To Select a Minimum Between them and
363              ↳ Continue Process! Breaking..."
364              , Alerts.Notice);
365         break;
366     }
367     char[] tempNodesConnectedTo = Get.TempNodesConnectedTo(currentNode);
368     if (tempNodesConnectedTo ≠ null)
369     {
370         //skip and redo current node with a new node in
371         ↳ NodesToSelectMinBetween
372         Output.Alert(
373             "Nodes Connected to Node " + currentNode + " are: " +
374             Output.Arrays.Char(tempNodesConnectedTo, false)
375             , Alerts.Info);
376         foreach (char node in tempNodesConnectedTo)
377         {
378             Output.Alert("chosenNode = " + node, Alerts.Info);
379             //Dijkstra: Here We Calculate the PathCost for each node
380             //PathCost = (CurrentNode's PathCost + the Cost of going to
381             ↳ new Node)
382             int pathCost_temp =
383                 _dijkstraNodes[currentNode - Lettercode - 1].PathCost +
384                 Get.DataFromArray.NodeTableInts(currentNode, node);
385             if (pathCost_temp < 0)
386             {
387                 error = "fragmental Network";
388                 break;
389             }
390             Output.Alert("totalPathCost = " + pathCost_temp, Alerts.Info);
391             //Dijkstra: If the calculated PathCost is less than previous
392             ↳ PathCost we update it with ...
393             //...new PathCost. Then we set a mark (ForeNode) that
394             ↳ reminds us how we got to this node
395             if (pathCost_temp < _dijkstraNodes[node - Lettercode -
396             ↳ 1].PathCost)
397             {
398                 Output.Alert("last cost = " + _dijkstraNodes[node -
399                 ↳ Lettercode - 1].PathCost,
400                 Alerts.Info);
401                 _dijkstraNodes[node - Lettercode - 1].PathCost =
402                 ↳ pathCost_temp;

```

```

۳۹۵         _dijkstraNodes[node - Lettercode - 1].ForeNode =
           ↳ currentNode;
۳۹۶         Output.Alert("UPDATE: value applied", Alerts.Info, false);
۳۹۷     }
۳۹۸     //Dijkstra: we keep re-doing these steps till we declare state
           ↳ of all nodes which are ...
۳۹۹         //...connected to the _sourceNode. then we process the
           ↳ Graph
۴۰۰     //Dijkstra: There are a bit difference in how Dijkstra
           ↳ proceeds the first time (nodes ...
۴۰۱         //... Connected to the _sourceNode), how it generally runs
           ↳ in graph and finally how ...
۴۰۲         //... it does it in last time... PLEASE pay attention to
           ↳ details of these Method (Run())
۴۰۳     }
۴۰۴ }
۴۰۵ else
۴۰۶ {
۴۰۷     Output.Alert
۴۰۸         ("CAUTION: Failed: Could NOT find any node ahead of this
           ↳ node...", Alerts.Error, false);
۴۰۹     Output.Alert("Retrying other node"
           ↳ , Alerts.Notice );
۴۱۰ }
۴۱۱
۴۱۲     //Dijkstra: Select Minimum Node in this graph to Set as new
           ↳ CurrentNode
۴۱۳     //Shift doesn't let program to do any loops if all values are same.
۴۱۴     //If there are two candidates for minChar first time it will select
           ↳ first, then next...
۴۱۵     char lastMinChar = minChar;
۴۱۶     minChar = SelectMinChar(nodesToSelectMinBetween, shift );
۴۱۷     if (!FirstTimeFor_sourceNode &&
۴۱۸         lastMinChar == minChar &&
۴۱۹         lastMinChar !=
           ↳ nodesToSelectMinBetween[nodesToSelectMinBetween.Length -1])
۴۲۰     {
۴۲۱         minChar = SelectMinChar(nodesToSelectMinBetween, ++shift);
۴۲۲     }
۴۲۳     else if (FirstTimeFor_sourceNode)
۴۲۴     {
۴۲۵         minChar = SelectMinChar(nodesToSelectMinBetween );
۴۲۶         FirstTimeFor_sourceNode = false;
۴۲۷     }
۴۲۸     else if (lastMinChar == minChar)

```

```

FF9         {
FF0             break;
FF1         }
FF2
FF3         currentNode = minChar;
FF4
FF5         Output.Alert("minchar = " + minChar, Alerts.Info);
FF6         //Dijkstra: when here: we restart the whole process to find the path
FF7         ↪ to _destinationNode ...
FF8             //... (when _destinationNode.Status = Permanent)
FF9         //In my app I process till all nodes go 'p' because that way we have
FF0         ↪ something named 'Shortest ...
FF1             //... Path Tree' which helps us to find any shortest path from
FF2         ↪ this constant start point. ...
FF3             //... In other words: My app draws a Shortest Path Tree from Start
FF4         ↪ point to everywhere then ...
FF5             //... Selects the destination and gets the path via the tree.
FF6         //TODO add a branch which this class is different: so program can draw the tree
FF7         ↪ and print it: no destination!
FF8         if (_dijkstraNodes[currentNode - Lettercode - 1].PathCost ≠ Infinite)
FF9             _dijkstraNodes[currentNode - Lettercode - 1].Status = 'p';
FF0         Output.Alert("currentNode's (" + currentNode + ") status = " +
FF1             _dijkstraNodes[currentNode - Lettercode - 1].Status
FF2             , Alerts.Info);
FF3         Output.BR();
FF4     }
FF5     Output.Alert("END: Dijkstra Process", Alerts.Notice);
FF6     Output.BR(5);
FF7
FF8     int pathCost = _dijkstraNodes[_destinationNode - Lettercode - 1].PathCost;
FF9     if (pathCost = Infinite || error = "fragmental Network")
FF0         Output.Alert("This Destination is not Accessable from this Start
FF1         ↪ Node", Alerts.Error);
FF2     else
FF3         Output.Alert("This is the cost of shortest path possible: " +
FF4             pathCost + " via path between these Nodes: " +
FF5             Output.Arrays.Char(FindForeNodes(_destinationNode),
FF6             ↪ false)
FF7             , Alerts.Success);
FF8     }
FF9 }
FF0
FF1 private class Output
FF2 {

```

```

F6Y      public static void BR(int number = 1)
F6A      {
F6Q          for (int i = 1; i ≤ number; i++)
F70              Console.WriteLine();
F71      }
F72      public class Visuals
F73      {
F7E          [SuppressMessage("ReSharper", "UnusedMethodReturnValue.Local")]
F7D          public class HeaderFooter
F7F          {
F80              public static string Start(string data, bool print = true)
F81              {
F82                  data = ("\n===== " +
F83                          "\n >>> START: " + data +
F84                          "\n===== \n");
F85                  if (print) Alert(data, Alerts.Notice, false);
F86                  return data;
F87              }
F88              public static string End(string data, bool print = true)
F89              {
F90                  data = ("\n===== " +
F91                          "\n >>> END: " + data +
F92                          "\n===== \n");
F93                  if (print) Alert(data, Alerts.Notice, false);
F94                  return data;
F95              }
F96          }
F97      }
F98      public class Table
F99      {
F9A          public static void Int(int[,] array)//prog*
F9B          {
F9C              string mode_temp = "Int Table Output";
F9D              HeaderFooter.Start(mode_temp);
F9E              Console.Write("\n");
F9F              int i;
F9G              for (i = 1; i < array.GetLength(0); i++)
F9H                  Console.Write(" | " + (char) array[i, 0] + " | ");
F9I              for (i = 1; i < array.GetLength(0) ; i++)
F9J              {
F9K                  Console.Write("\n | " + (char) array[i, 0] + " |");
F9L                  for (int j = 1; j < array.GetLength(1); j++)
F9M                      Console.Write(" | {0, 3}", array[i, j]);
F9N              }
F9O              BR();
F9P              HeaderFooter.End(mode_temp);

```

```

012     }
013 }
014 }
015 public class Arrays
016 {
017     public static string Char(char[] array, bool print = true, bool error
    ↪ = Error)
018     {
019         if (array == null)
020         {
021             if (error) Alert("Null Array to visualize!", Alerts.Error);
022             return "";
023         }
024         string data = ("{" + array[0]);
025         int i = 1;
026         if (array.Length == 1)
027             data += ("}");
028         else if (array.Length > 1)
029         {
030             data += (", ");
031             if (array.Length ≥ 3)
032                 for (; i < array.GetLength(0) - 1; i++)
033                     data += (array[i] + ", ");
034             data += (array[i] + "}");
035         }
036         if (print) Console.WriteLine(data);
037         return data;
038     }
039 }
040 public static void Alert (string message, Alerts type = Alerts.Default, bool
    ↪ badge = true)
041 {
042     switch (type)
043     {
044         case Alerts.Error:
045             Console.ForegroundColor = ConsoleColor.DarkRed;
046             if (badge) Console.Write("ERROR: ");
047             Console.WriteLine(message);
048             Console.ForegroundColor = ConsoleColor.Gray;
049             break;
050
051         case Alerts.Info:
052             Console.ForegroundColor = ConsoleColor.Blue;
053             if (badge) Console.Write("INFO: ");
054             Console.WriteLine(message);

```

```

0505         Console.ForegroundColor = ConsoleColor.Gray;
0506         break;
0507
0508     case Alerts.Notice:
0509         Console.ForegroundColor = ConsoleColor.DarkYellow;
0510         if (badge) Console.Write("NOTICE: ");
0511         Console.WriteLine(message);
0512         Console.ForegroundColor = ConsoleColor.Gray;
0513         break;
0514
0515     case Alerts.Success:
0516         Console.ForegroundColor = ConsoleColor.Green;
0517         if (badge) Console.Write("SUCCESS: ");
0518         Console.WriteLine(message);
0519         Console.ForegroundColor = ConsoleColor.Gray;
0520         break;
0521     case Alerts.DefaultWithBR:
0522         Console.WriteLine(message);
0523         break;
0524     //case Alerts.Default://redundant
0525     default:
0526         Console.Write(message);
0527         break;
0528 }
0529 }
0530 }
0531 private class Input
0532 {
0533     public static string String(string message = "Enter a Text String: "
0534         , Alerts alertType = Alerts.Default
0535         , bool reports = Reportavailable)
0536     {
0537         Output.Alert(message, alertType);
0538         string value = Console.ReadLine();
0539         if (!reports) return value;
0540         switch (value)
0541         {
0542             case "-1":
0543                 Output.Visuals.Table.Int(_nodesTableInts);
0544                 break;
0545         }
0546         return value;
0547     } //Handwrite/Hardcore*
0548     public static string Bot(string message = "", bool printValue = true, bool
0549         ↪ addNth = true)

```

```

099 {
100     Console.Write(message);
101     string value = Get.StringsNthpart(_stringNthRequired);
102     if (addNth) _stringNthRequired++;
103     if (printValue) Output.Alert("BOT INPUT: " + value, Alerts.Notice);
104     Output.BR();
105     return value;
106 }
107 public static int Int32(string message = "Enter an Integer: "
108     , int min = int.MinValue
109     , int max = int.MaxValue
110     , bool error = Error
111     , bool checkReportPossible = Reportavailable)
112 {
113     if (max < min)
114     {
115         int _temp = min;
116         min = max;
117         max = _temp;
118     }
119     int value;
120     do
121     {
122         Console.Write(message);
123         string inputString = String("", reports: checkReportPossible);
124         if (!string.IsNullOrEmpty(inputString))
125         {
126             bool negativeFlag = inputString.Contains("-");
127             inputString = Regex.Match(inputString, @"^\d+").Value;
128             if (!string.IsNullOrEmpty(inputString))
129             {
130                 value = int.Parse(inputString);
131                 if (negativeFlag) value = -value;
132                 if (min > value && error)
133                     Output.Alert("Your input is so small. Minimum is " + min +
134                         ↵ ".", Alerts.Error);
135                 else
136                     if (max < value && error)
137                         Output.Alert("Your input is so big. Maximum is " + max +
138                             ↵ ".", Alerts.Error);
139                 else break;
140             }
141             if (error) Output.Alert("Invalid input", Alerts.Error);
142         }
143         else if (error) Output.Alert("Empty input", Alerts.Error);

```

```

642         } while (true);
643         return value;
644     }
645     public static char Char(string message = "Enter a Character: ")
646         , char min = 'A'
647         , char max = 'Z'
648         , bool error = Error
649         , bool checkReportPossible = Reportavailable)
650     {
651         char value;
652         do
653         {
654             Console.Write(message);
655             string inputString = String("", reports: checkReportPossible);
656             if (!string.IsNullOrEmpty(inputString))
657             {
658                 if (inputString.Length > 1)
659                 {
660                     if (error) Output.Alert("Only 1 character input",
661                                             ↳ Alerts.Error);
662                 }
663                 else
664                 {
665                     value = char.Parse(inputString.ToUpper());
666                     if (min > value && error)
667                         Output.Alert("Your input is irregular. start is " + min +
668                                     ↳ ".", Alerts.Error);
669                     else if (max < value && error)
670                         Output.Alert("Your input is irregular. end is " + max +
671                                     ↳ ".", Alerts.Error);
672                     else break;
673                 }
674             }
675             else if (error) Output.Alert("Empty input", Alerts.Error);
676         } while (true);
677         return value;
678     }
679 }
680
681 private class Get
682 {
683     public class DataFromArray
684     {
685         public static int NodeTableInts(char x, char y)//prog
686         {
687             y = char.Parse(y.ToString().ToUpper());

```



```

688F         x = char.Parse(x.ToString().ToUpper());
6890         return _nodesTableInts[x - Lettercode, y - Lettercode];
6891     }
6892 }
6893
6894
6895
6896
6897
6898
6899
6900
6901
6902
6903
6904
6905
6906
6907
6908
6909
6910
6911
6912
6913
6914
6915
6916
6917
6918
6919
6920
6921
6922
6923
6924
6925
6926
6927
6928
6929
6930
6931
6932
6933
6934
6935
6936
6937
6938
6939
6940
6941
6942
6943
6944
6945
6946
6947
6948
6949
6950
6951
6952
6953
6954
6955
6956
6957
6958
6959
6960
6961
6962
6963
6964
6965
6966
6967
6968
6969
6970
6971
6972
6973
6974
6975
6976
6977
6978
6979
6980
6981
6982
6983
6984
6985
6986
6987
6988
6989
6990
6991
6992
6993
6994
6995
6996
6997
6998
6999
7000
7001
7002
7003
7004
7005
7006
7007
7008
7009
7010
7011
7012
7013
7014
7015
7016
7017
7018
7019
7020
7021
7022
7023
7024
7025
7026
7027
7028
7029
7030
7031
7032
7033
7034
7035
7036
7037
7038
7039
7040
7041
7042
7043
7044
7045
7046
7047
7048
7049
7050
7051
7052
7053
7054
7055
7056
7057
7058
7059
7060
7061
7062
7063
7064
7065
7066
7067
7068
7069
7070
7071
7072
7073
7074
7075
7076
7077
7078
7079
7080
7081
7082
7083
7084
7085
7086
7087
7088
7089
7090
7091
7092
7093
7094
7095
7096
7097
7098
7099
7100
7101
7102
7103
7104
7105
7106
7107
7108
7109
7110
7111
7112
7113
7114
7115
7116
7117
7118
7119
7120
7121
7122
7123
7124
7125
7126
7127
7128
7129
7130
7131
7132
7133
7134
7135
7136
7137
7138
7139
7140
7141
7142
7143
7144
7145
7146
7147
7148
7149
7150
7151
7152
7153
7154
7155
7156
7157
7158
7159
7160
7161
7162
7163
7164
7165
7166
7167
7168
7169
7170
7171
7172
7173
7174
7175
7176
7177
7178
7179
7180
7181
7182
7183
7184
7185
7186
7187
7188
7189
7190
7191
7192
7193
7194
7195
7196
7197
7198
7199
7200
7201
7202
7203
7204
7205
7206
7207
7208
7209
7210
7211
7212
7213
7214
7215
7216
7217
7218
7219
7220
7221
7222
7223
7224
7225
7226
7227
7228
7229
7230
7231
7232
7233
7234
7235
7236
7237
7238
7239
7240
7241
7242
7243
7244
7245
7246
7247
7248
7249
7250
7251
7252
7253
7254
7255
7256
7257
7258
7259
7260
7261
7262
7263
7264
7265
7266
7267
7268
7269
7270
7271
7272
7273
7274
7275
7276
7277
7278
7279
7280
7281
7282
7283
7284
7285
7286
7287
7288
7289
7290
7291
7292
7293
7294
7295
7296
7297
7298
7299
7300
7301
7302
7303
7304
7305
7306
7307
7308
7309
7310
7311
7312
7313
7314
7315
7316
7317
7318
7319
7320
7321
7322
7323
7324
7325
7326
7327
7328
7329
7330
7331
7332
7333
7334
7335
7336
7337
7338
7339
7340
7341
7342
7343
7344
7345
7346
7347
7348
7349
7350
7351
7352
7353
7354
7355
7356
7357
7358
7359
7360
7361
7362
7363
7364
7365
7366
7367
7368
7369
7370
7371
7372
7373
7374
7375
7376
7377
7378
7379
7380
7381
7382
7383
7384
7385
7386
7387
7388
7389
7390
7391
7392
7393
7394
7395
7396
7397
7398
7399
7400
7401
7402
7403
7404
7405
7406
7407
7408
7409
7410
7411
7412
7413
7414
7415
7416
7417
7418
7419
7420
7421
7422
7423
7424
7425
7426
7427
7428
7429
7430
7431
7432
7433
7434
7435
7436
7437
7438
7439
7440
7441
7442
7443
7444
7445
7446
7447
7448
7449
7450
7451
7452
7453
7454
7455
7456
7457
7458
7459
7460
7461
7462
7463
7464
7465
7466
7467
7468
7469
7470
7471
7472
7473
7474
7475
7476
7477
7478
7479
7480
7481
7482
7483
7484
7485
7486
7487
7488
7489
7490
7491
7492
7493
7494
7495
7496
7497
7498
7499
7500
7501
7502
7503
7504
7505
7506
7507
7508
7509
7510
7511
7512
7513
7514
7515
7516
7517
7518
7519
7520
7521
7522
7523
7524
7525
7526
7527
7528
7529
7530
7531
7532
7533
7534
7535
7536
7537
7538
7539
7540
7541
7542
7543
7544
7545
7546
7547
7548
7549
7550
7551
7552
7553
7554
7555
7556
7557
7558
7559
7560
7561
7562
7563
7564
7565
7566
7567
7568
7569
7570
7571
7572
7573
7574
7575
7576
7577
7578
7579
7580
7581
7582
7583
7584
7585
7586
7587
7588
7589
7590
7591
7592
7593
7594
7595
7596
7597
7598
7599
7600
7601
7602
7603
7604
7605
7606
7607
7608
7609
7610
7611
7612
7613
7614
7615
7616
7617
7618
7619
7620
7621
7622
7623
7624
7625
7626
7627
7628
7629
7630
7631
7632
7633
7634
7635
7636
7637
7638
7639
7640
7641
7642
7643
7644
7645
7646
7647
7648
7649
7650
7651
7652
7653
7654
7655
7656
7657
7658
7659
7660
7661
7662
7663
7664
7665
7666
7667
7668
7669
7670
7671
7672
7673
7674
7675
7676
7677
7678
7679
7680
7681
7682
7683
7684
7685
7686
7687
7688
7689
7690
7691
7692
7693
7694
7695
7696
7697
7698
7699
7700
7701
7702
7703
7704
7705
7706
7707
7708
7709
7710
7711
7712
7713
7714
7715
7716
7717
7718
7719
7720
7721
7722
7723
7724
7725
7726
7727
7728
7729
7730
7731
7732
7733
7734
7735
7736
7737
7738
7739
7740
7741
7742
7743
7744
7745
7746
7747
7748
7749
7750
7751
7752
7753
7754
7755
7756
7757
7758
7759
7760
7761
7762
7763
7764
7765
7766
7767
7768
7769
7770
7771
7772
7773
7774
7775
7776
7777
7778
7779
7780
7781
7782
7783
7784
7785
7786
7787
7788
7789
7790
7791
7792
7793
7794
7795
7796
7797
7798
7799
7800
7801
7802
7803
7804
7805
7806
7807
7808
7809
7810
7811
7812
7813
7814
7815
7816
7817
7818
7819
7820
7821
7822
7823
7824
7825
7826
7827
7828
7829
7830
7831
7832
7833
7834
7835
7836
7837
7838
7839
7840
7841
7842
7843
7844
7845
7846
7847
7848
7849
7850
7851
7852
7853
7854
7855
7856
7857
7858
7859
7860
7861
7862
7863
7864
7865
7866
7867
7868
7869
7870
7871
7872
7873
7874
7875
7876
7877
7878
7879
7880
7881
7882
7883
7884
7885
7886
7887
7888
7889
7890
7891
7892
7893
7894
7895
7896
7897
7898
7899
7900
7901
7902
7903
7904
7905
7906
7907
7908
7909
7910
7911
7912
7913
7914
7915
7916
7917
7918
7919
7920
7921
7922
7923
7924
7925
7926
7927
7928
7929
7930
7931
7932
7933
7934
7935
7936
7937
7938
7939
7940
7941
7942
7943
7944
7945
7946
7947
7948
7949
7950
7951
7952
7953
7954
7955
7956
7957
7958
7959
7960
7961
7962
7963
7964
7965
7966
7967
7968
7969
7970
7971
7972
7973
7974
7975
7976
7977
7978
7979
7980
7981
7982
7983
7984
7985
7986
7987
7988
7989
7990
7991
7992
7993
7994
7995
7996
7997
7998
7999
8000

```

```

Y29     }
Y30
Y31     [SuppressMessage("ReSharper", "UnusedMember.Local")]
Y32     public static bool UniqueCharsState(string data) //Extra
Y33     {
Y34         if(data.Length > 256)
Y35             return false;
Y36         bool[] char_set = new bool[256];
Y37         foreach (int val in data)
Y38         {
Y39             if(char_set[val])
Y40                 return false;
Y41             char_set[val] = true;
Y42         }
Y43         return true;
Y44     }
Y45
Y46     public static char[] TempNodesConnectedTo(char thisNode)
Y47     {
Y48         string data = "";
Y49         for (int i = 1; i ≤ _nodesCount; i++)
Y50         {
Y51             int nodeTableReturn = DataFromArray.NodeTableInts(thisNode, (char)(i +
Y52                 ↵ Lettercode));
Y53
Y54             if (nodeTableReturn > 0 && _dijkstraNodes[i - 1].Status == 't')
Y55                 data += (char)(i + Lettercode);
Y56         }
Y57
Y58         if (data.Length == 0)
Y59             return null;
Y60         char[] nodesConnectedToSource = new char[data.Length];
Y61         int c = 0;
Y62         foreach (char cha in data)
Y63         {
Y64             nodesConnectedToSource[c] = cha;
Y65             c++;
Y66         }
Y67         return nodesConnectedToSource;
Y68     }
Y69
Y70     public static char[] CharArrayFromString(string data)
Y71     {
Y72         if (data.Length == 0)
Y73             return null;

```

```

Y7T         char[] CharData = new char[data.Length];
Y7F         int c = 0;
Y7D         foreach (char cha in data)
Y7E         {
Y7Y             CharData[c] = cha;
Y7A             c++;
Y79         }
Y7.-        return CharData;
Y7A1       }

Y7AT       public static string StringFromCharArray(char[] array)
Y7AF       {
Y7AD           string value = "";
Y7AF           if (array != null)
Y7AY               foreach (char cha in array)
Y7AA                   value += cha;
Y7A9           return value;
Y7.-       }
Y7A1     }
Y7A2     private class Job
Y7A3     {
Y7AF         [SuppressMessage("ReSharper", "UnusedMember.Local")]
Y7AD         private static string ConvertIntToAnyString(int value, int cap) //Extra / For
Y7D-         ↪ Update*
Y7D-         {
Y7DY             string result = "";
Y7DA             char[] baseChars = new char[cap];
Y7D9             for (int i = 0; i < cap; i++)
Y7D.-                 baseChars[i] = (char)(Lettercode + i);
Y7D-             int targetBase = baseChars.Length;
Y7D-             do
Y7D-             {
Y7D-                 char letter = baseChars[value % targetBase];
Y7D-                 if (letter == '@') letter = '0';
Y7D-                 result = letter + result;
Y7D-                 value = value / targetBase;
Y7D-             }
Y7D-             while (value > 0);
Y7D-             return result;
Y7D1         }

Y7D2       public static string ExtractNumbers(string data)
Y7D3       {
Y7D5           int value = 0;
Y7D6           if (!string.IsNullOrEmpty(data))

```

```

817         {
818             bool negativeFlag = data.Contains("-");
819             data = Regex.Match(data, @"\d+").Value;
820             if (string.IsNullOrEmpty(data)) return value.ToString();
821             value = int.Parse(data);
822             if (negativeFlag) value = -value;
823         }
824         else return null;
825         return value.ToString();
826     }
827
828     public static bool CheckNodeConnection(char node)
829     {
830         for(int i = 1; i ≤ _nodesCount; i++)
831             if (Get.DataFromArray.NodeTableInts(node, (char) (i + Lettercode)) >
832                 0)
833                 return true;
834         return false;
835     }
836 }
837 }

```