مستند مراحل کلی توسعهٔ نرم‌افزار 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 توسعهٔ پروژهٔ سال دوم هنرستانم (۹۷-۹۶) — پیدا کردن بهینه‌ترین مسیر در شبکه — در رشتهٔ نرم‌افزار و شبکه است.

از آن جهت که به عنوان یک دانش‌آموز هنرستانی پیش از آن هرگز با چنین مسئله‌ای روبرو نشده بودم؛ می‌دانستم که هر اشتباه در مسیر توسعه می‌تواند به هزینهٔ هنگفتی در زمان منجر شود. از همین رو مطالب مهم را در سندی تألیف کردم.

نسخهٔ پیش رو ویرایشی از سند اصلی است که در TEX باز نویسی شده است. این نسخه همچنین شامل بخش توضیحات اولیه و مفاهیم نیز می‌باشد.

توجه کنید که این کد و توضیحات قدیمی می‌باشند و کیفیتی مدرسه‌ای دارند! این سند صرفاً بازنویسی یادداشت‌های اصلی است. کد در انتهای سند است. (پ.ن: کیفیت کد به گونه‌ای است که من به عنوان نویسنده اصلی هم نمی‌توانم از آن استفاده کنم.)

فهرست مطالب

۱ مفاهیم

قبل از درک نرم‌افزار نیاز است مفاهیم پایهٔ گراف‌ها را به خوبی بیاموزید. مطالب زیر برداشت‌هایی از Peter Linz در An introduction to Formal Languages and Automata است.

هر **گراف** ساختاری است که از دو مجموعه منتهی نقاط (Vertices) یا گره‌ها (Nodes) و یال‌ها (Edges) تشکیل می‌شود.

برای نمایش این اتصالات از **دیاگرام**‌های متفاوتی استفاده می‌شود؛ این دیاگرام‌ها غالباً از دایره برای نمایش نقاط، خطوط یا پیکان‌ها برای نمایش یال‌های جهت‌دار و بی‌جهت استفاده می‌کنند و مقیاس ندارند. مقیاس نداشتن به این معنا است که طول هر خط ارتباطی با طول یا وزن حقیقی‌اش ندارد و فقط نمایانگر اتصال است نه طول خط.

هر **یال (Edge)** از دو نود تشکیل می‌شود. به طور مثال: $e_i = (v_j, v_k)$ یالی است که از v_j به سمت v_k می‌رود به این معنا که جهتی یک طرفه به سمت v_j دارد به چنین یالی **کمان (Arc)** نیز می‌گویند. اگر که در تعریف هر گراف آمده باشد که گراف جهت‌دار یا هدایت‌شده (Directed) است، نمی‌توان دیگر در این دایگراف (Digraph) از v_k به v_j برگشت؛ مگر اینکه گراف جهت‌دار نباشد یا گراف از نوع مخلوط باشد — نوعی که شامل یال‌های دوطرفه و یک‌طرفه 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 یال‌ها می‌توانند وزن یا به بیان دیگر هزینهٔ پیمایش داشته باشند. موزون بودن گراف باعث می‌شود که مفهوم «طول یال» معنا پیدا کند.

هر **پیمایش (Walk)** ترتیبی از یال‌ها به صورت $(v_i, v_j), (v_j, v_k), \ldots, (v_m, v_n)$ از v_i تا v_n است. در شبکه‌ها طول این پیمایش به اندازهٔ جمع هزینه‌های یال‌ها است. در گراف‌ها این مقدار به اندازهٔ تعداد یال‌ها است چرا که در اثر عدم وجود وزن، هر یال هزینه‌ای واحد (۱) دارد.

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

مسیر ساده (Simple) مسیری است که در آن گذر از هیچ یالی تکرار نشود. از آنجایی که نرم‌افزار نهایی از حالات محاسباتی متفاوت و پیچیده‌ای پشتیبانی نمی‌کند، در تمام طول این سند (به جز این بخش) «مسیر» خلاصه‌ای 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 می‌شوند **برچسب (Label)** نامیده می‌شوند. مانند نام نودها یا وزن یال‌ها.

در نهایت، یالی از یک نود به خودش $e_l = (v_i, v_i)$ یک **حلقه (Loop)** خوانده می‌شود.

۲ اطلاعات اولیه

در زبان C# برنامه‌ای بنویسید که بهینه‌ترین مسیر ساده را در یک شبکه بی‌جهت پیدا کند.

زمان شروع: ۲۰۱۸/۳/۲۰ (۱۳۹۶/۱۲/۲۹)

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

۳ یادداشت‌های اولیه

۱.۳ یادداشت اولیه

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

- هر نود رشته‌ای اختصاصی دارد که نشان می‌دهد به چه نقاط دیگری وصل است (a ← cd)
- هر نود آرایه‌ای دارد که هزینه جابه‌جایی 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 ۲ × ۲ به شکل جدولی از اطلاعات با سطر و ستون‌هایی به اندازهٔ تعداد نودهای گرفته شده از مرحلهٔ 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 ← 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.

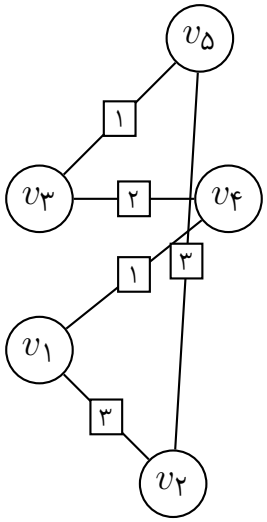
تودو

□ 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.

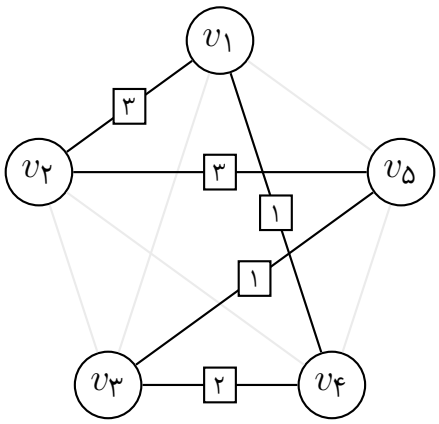
۱.۲.۳ توضیحات

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) رامرتب 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 + قطرها) به واسطه نقاط است. $|Edges| = \frac{nodes \times (nodes-۱)}{۲}$



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 نقطهٔ آغازین یک یال است و هر ستون نمایندهٔ یک نقطهٔ پایانی برای همان یال است.

×	$v_۱$	$v_۲$	$v_۳$	$v_۴$	$v_۵$
$v_۱$					
$v_۲$					
$v_۳$					
$v_۴$					
$v_۵$					

جدول ۱: نمونه جدول مشخصات یک شبکه مفروض

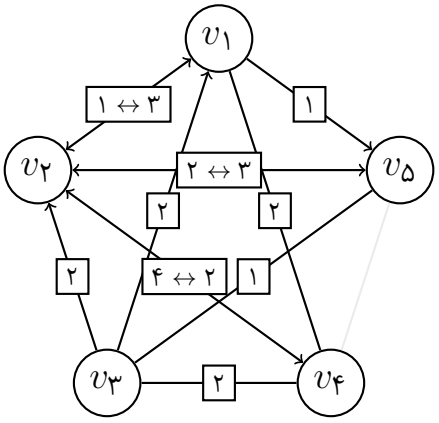
علت غیرفعال بودن قطر جداول آن است که پشتیبانی از حلقه‌ها در برنامه وجود ندارد. به همین دلیل نمی‌توان از یک نقطه به همان نقطه رفت.

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

×	$v_۱$	$v_۲$	$v_۳$	$v_۴$	$v_۵$
$v_۱$		۳	۰	۱	۰
$v_۲$	۳		۰	۰	۳
$v_۳$	۰	۰		۲	۱
$v_۴$	۱	۰	۲		۰
$v_۵$	۰	۳	۱	۰	

جدول ۲: اطلاعات پیاده شدهٔ شبکه‌های شکل‌های ۱ و ۲

نکتهٔ 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

×	$v_۱$	$v_۲$	$v_۳$	$v_۴$	$v_۵$
$v_۱$		۱	۰	۲	۱
$v_۲$	۳		۰	۲	۳
$v_۳$	۲	۲		۲	۱
$v_۴$	۲	۴	۲		۰
$v_۵$	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 نسخهٔ اصلی این سند، تا به اینجای کار از امکان وجود مسیرهای منفی در شبکه‌ها اطلاعی نداشته‌ام و در ادامهٔ این کاوش، هنگامی که به انتخاب الگوریتم خواهم رسید، بر حسب اتفاق الگوریتمی را انتخاب خواهم 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 محاسبهٔ تمام مسیرها بر اساس مبدأ مورد نظر به انتخاب مقصد بپردازیم.

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 مبادا نقشهٔ فواصل محلهٔ جدید خود را به خاطر بسپارید تا اگر گاهی دیرتان شد بدانید بهترین مسیر کدام است (بهینه‌ترین مسیر). نقشهٔ محلهٔ 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 (محاسبهٔ 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 خانهٔ 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 پیش‌ـپردازش همهٔ مسیرهای (درخت بهینه‌ترین مسیر) هر شبکه:

۱. محاسبهٔ همهٔ مسیرها به نسبت مبدأ

۲. تعیین مقصد

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

۱.۴ روش‌اولیه (منطقی یا انسانی)

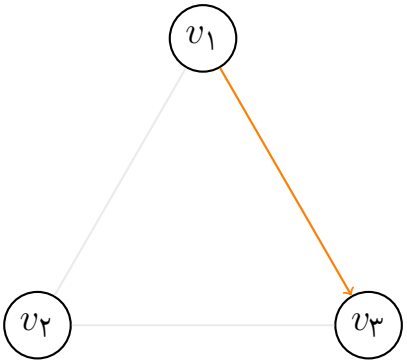
تنها روشی که اوایل ۲۰۱۸/۳ (۱۳۹۶/۱۲) به ذهنم رسید به این صورت بود که اولین کاری که باید برسانم دانستن تعداد تمام مسیرهای ممکن است. پس از آن محاسبهٔ وزن هر مسیر و مقایسهٔ آنهاست. از آنجا که این روش بر اصول یا اثبات ریاضی تأکیدی ندارد و صرفاً اولین راه حل منطقی‌ام برای حل مسئله بود؛ آنرا روش انسانی نامیده‌ام.

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 مِش) 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)



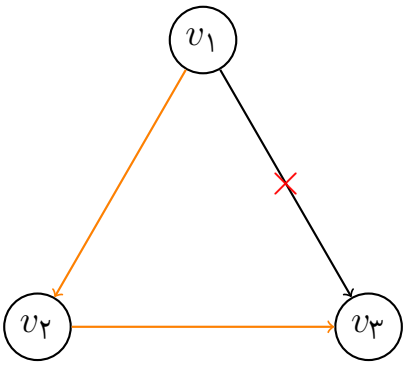
(ب) گراف مخلوط نمونه

(آ) لیست مسیرها

۱	$v_۱, v_۲$

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 همهٔ مسیرها به دست آورده شوند.
اگر بخواهیم از $v_۱$ به $v_۳$ برویم؛ کوتاه‌ترین مسیر مستقیم دو نودی است: $v_۱, v_۲$
در شکل ۴ این مسیر را لیست کرده‌ایم. حال این کار را انقدر تکرار می‌کنیم تا تمام مسیرهای دو نودی که از نود اول ($v_۱$) 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 همهٔ نودها همهٔ مسیرهای یکتای تک یالی را رفته باشند.



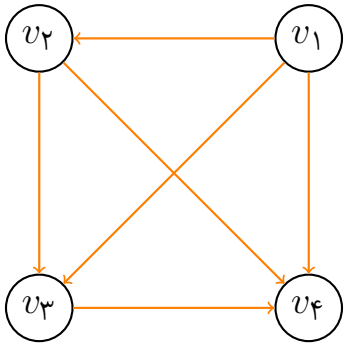
(ب) گراف مخلوط نمونه

(آ) لیست کامل مسیرها

۱	$v_۱, v_۲$
۲	$v_۱, v_۲, v_۳$

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 می‌نویسم: $v_۱, v_۴$
سپس مسیرهای سه نودی را بررسی می‌کنیم. (جدول ۳)

۱	$v_۱, v_۲, v_۴$
۲	$v_۱, v_۳, v_۴$

جدول ۳: لیست مسیرهای سه نودی ممکن از $v_۱$ به $v_۳$

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.

۱	$v_۱, v_۲, v_۳, v_۴$
۲	$v_۱, v_۳, v_۲, v_۴$

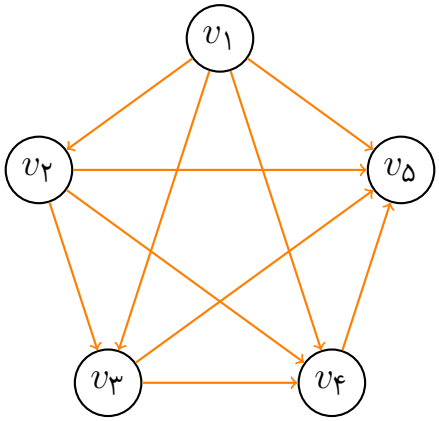
جدول ۴: لیست مسیرهای چهار نودی ممکن از $v_۱$ به $v_۴$

حال که مسیرهای چهار نودی نیز لیست شده‌اند حداکثر حالات ممکنی که می‌توان از $v_۱$ به $v_۴$ رسید به شرح زیر است.

۱	$v_۱, v_۴$
۲	$v_۱, v_۲, v_۴$
۳	$v_۱, v_۳, v_۴$
۴	$v_۱, v_۲, v_۳, v_۴$
۵	$v_۱, v_۳, v_۲, v_۴$

جدول ۵: لیست کامل مسیرهای چهار نودی

باز از سر، برای به دست آوردن حداکثر حالاتی که از $v_۱$ به $v_۵$ می‌توان به رفت؛ به همین صورت عمل کرد و ابتدا با مسیر دو نودی مستقیم از $v_۱$ به $v_۵$ و سپس با اضافه کردن نود و باز محاسبه، کار را به پایان رساند.



شکل ۷: گراف پنج نودی مخلوط نمونه

۱	$v_۱, v_۵$
۲	$v_۱, v_۲, v_۵$
۳	$v_۱, v_۳, v_۵$
۴	$v_۱, v_۴, v_۵$
۵	$v_۱, v_۲, v_۳, v_۵$
۶	$v_۱, v_۲, v_۴, v_۵$
۷	$v_۱, v_۳, v_۲, v_۵$
۸	$v_۱, v_۳, v_۴, v_۵$
۹	$v_۱, v_۴, v_۲, v_۵$
۱۰	$v_۱, v_۴, v_۳, v_۵$
۱۱	$v_۱, v_۲, v_۳, v_۴, v_۵$
۱۲	$v_۱, v_۲, v_۴, v_۳, v_۵$
۱۳	$v_۱, v_۳, v_۲, v_۴, v_۵$
۱۴	$v_۱, v_۳, v_۴, v_۲, v_۵$
۱۵	$v_۱, v_۴, v_۲, v_۳, v_۵$
۱۶	$v_۱, v_۴, v_۳, v_۲, v_۵$

جدول ۶: لیست کامل مسیرهای پنج نودی ممکن از $v_۱$ به $v_۵$

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. $(Base = |Vertices| + ۱)$

به طور مثال اگر گراف مفروض ۵ نود داشته باشد. سامانهٔ 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 مقایسهٔ 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

$[s][Variant][d]$

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 مرحلهٔ 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 مرحلهٔ ۹ بازگرد.

ii. در غیر این صورت به مرحله بعد برو.

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

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

۲.۱.۴ تجربه با روش انسانی

با نوشتن برنامهٔ 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 رسیدم 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 دربارهٔ 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 ادامهٔ 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 دوباه‌ـطراحی کنم و اینبار تنها به دانش ناقص خودم تکیه نخواهم کرد و از منابع آزاد برداشت‌هایی خواهم کرد.

پس از بررسی‌هایی سطی دربارهٔ الگوریتم‌های حلال مسئله‌های «کوتاه‌ترین مسیرها تک‌ـمنبعی/آغازی» (Single-Source Shortest Paths) 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:

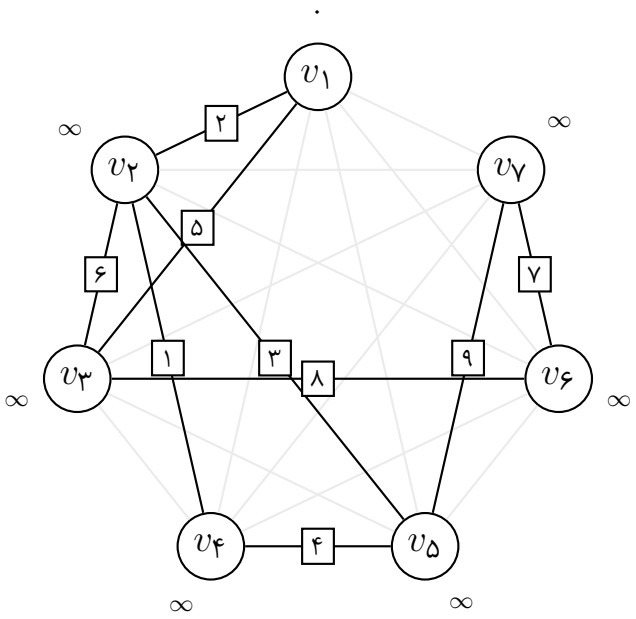
- الگوریتم Bellman–Ford
  - با تأکید بر بروزرسانی و پیمایش چندین و چند بارهٔ تمام شبکه
  - برای پیدا کردن کوتاه‌ترین مسیر در شبکه‌هایی که شامل چرخه‌های منفی نیست.
- الگوریتم Dijkstra
  - با تأکید بر افزایش بهینگی و دقت در مقایسه با الگوریتم Bellman–Ford
  - برای پیدا کردن کوتاه‌ترین مسیر در شبکه‌هایی 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 حوزهٔ 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 $v_۱$ به شهر $v_۸$ 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 نقطهٔ شروع تا آن نقطه چه مقدار است؟

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

۳. تابحال این این نقطه بررسی شده است؟

در فرض اولیه برای شروع کار الگوریتم:

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

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

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

با داشتن مفروضات اولیه الگوریتم می‌تواند شروع به کار کند.

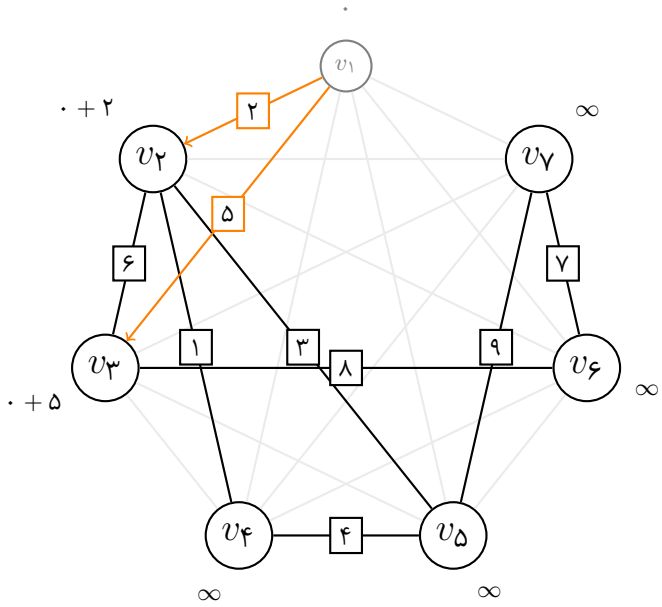
فاصلهٔ تمام یال‌هایی که از مبدأ قابل دسترسی هستند را می‌گیریم. (مثال: $v_۱, v_۲$).

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.

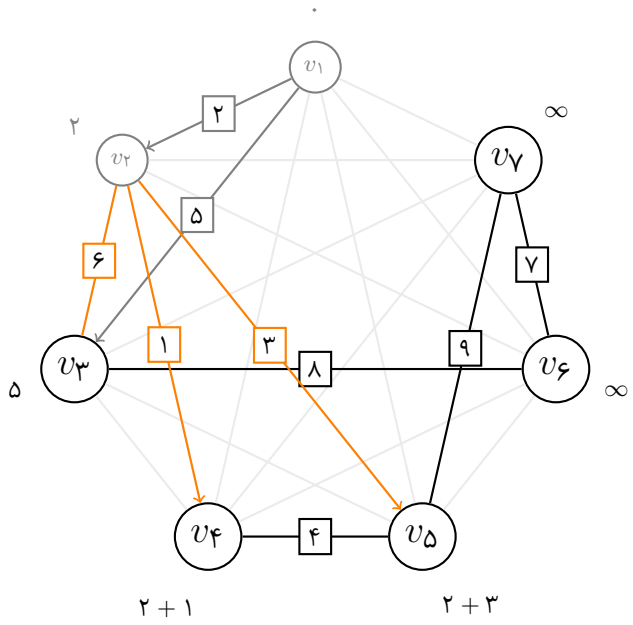
۲. اگر تمام نودهای متصل به مبدأ بررسی شدند. نود مبدأ را «بررسی شده» در نظر می‌گیریم.

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 همهٔ 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 $v_۲$ — انتخاب می‌کنیم.



شکل ۱۰: گراف نمونه که در ادامهٔ بررسی، از نودی که کمترین هزینه را داشته است ($v_۲$بین $v_۲$ و $v_۳$) فرآیند بررسی و برچسب‌گذاری را دوباره انجام می‌دهیم.

پس از انتخاب نود بعدی از بین نودهای متصل به نود فعلی ($v_۲$) از بین نودهای متصل به $v_۱$ یعنی $(v_۱, v_۲)$ شروع به تکرار فرآیند پیشتر ذکر شده می‌کنیم که در زیر دوباره تشریح شده است. (مطابق شکل ۱۰)

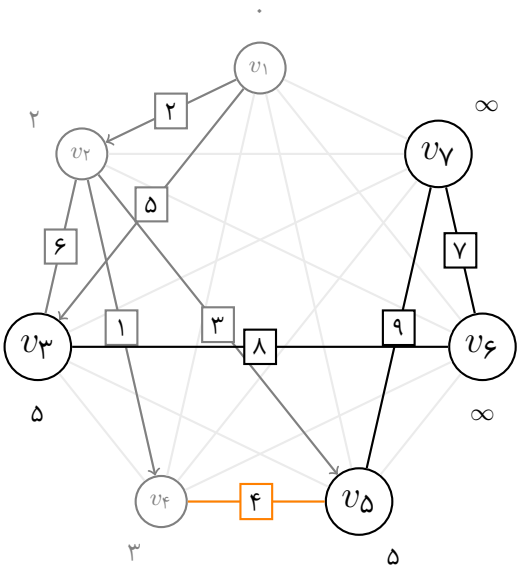
۱. بررسی می‌کنیم که از $v_۲$ به چه نودهایی دسترسی داریم؟ $(v_۳, v_۴, v_۵)$

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

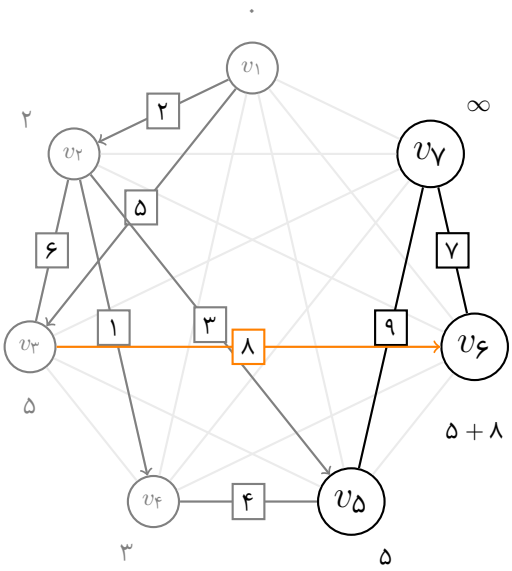
(آ) اگر طول مسیری که از پدر به آن نود می‌رسد کمتر از برچسب مسیر سابق بود آنرا به روز رسانی می‌کنیم. با بروزرسانی این برچسب، برچسب «نود پدر» را هم بروزرسانی می‌کنیم.
(در اولین مثال: آیا ۶ + ۲ از هزینهٔ قبلی نود (۵) کمتر است؟ چون جواب خیر است کاری به نود نداریم.)
(در دومین مثال: آیا ۱ + ۲ از هزینهٔ قبلی نود (∞) کمتر است؟ چون جواب بله است برچسب قدیمی (∞) حذف می‌شود و هزینهٔ جدید را ثبت می‌کنیم. همچنین یادداشت می‌کنیم که پدر $v_۴$ نود $v_۲$ است.)

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

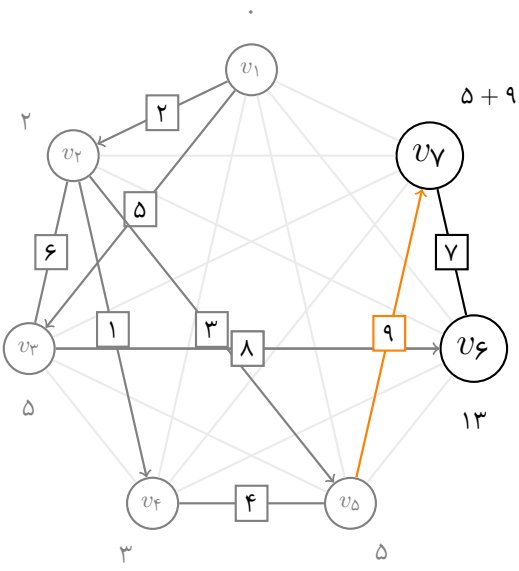
تمام این پروسه را آنقدر تکرار می‌کنیم تا همهٔ نودها به حالت «بررسی شده» درآیند.



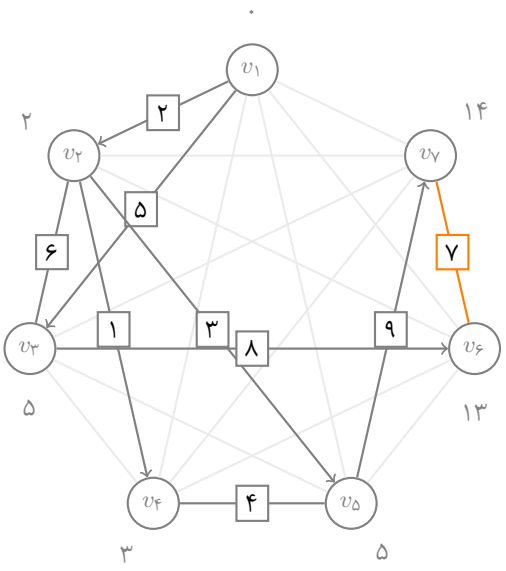
شکل ۱۱: گراف نمونه در ادامهٔ بررسی



شکل ۱۲: گراف نمونه در ادامهٔ بررسی

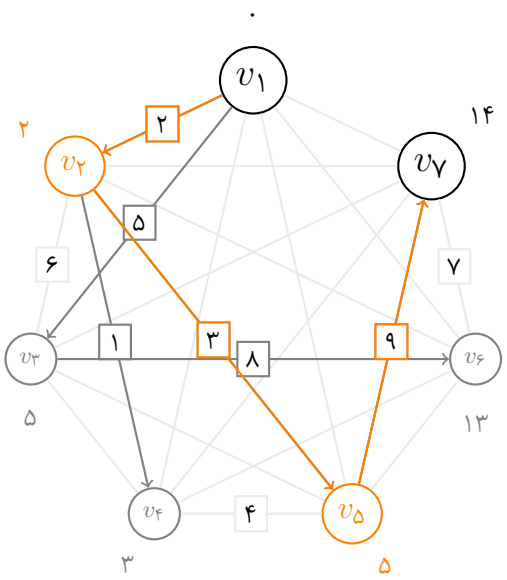


شکل ۱۳: گراف نمونه در ادامهٔ بررسی



شکل ۱۴: گراف نمونه در آخرین مرحلهٔ بررسی

هنگامی که همهٔ 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.

Code ۶

```
// By M. Yas. Davoodeh
// This is my first program written in C# as a highschool project.
// THE QUALITY OF IT IS NOT ACCEPTABLE! NOTICE AGAIN: THIS IS A HIGHSCHOOL LEVEL CODE!
/*
Some notes:

Handwriten/Hardcore Codes: codes which are in noobish style and not flexible.
They are so easy to break. Check every single one of them if you make any change to the
↪  program.

Modes: This little program supports two modes of inputs. One (Default) helps you
with creating a graph in peace and breaks relatively hardly (for users). The Second one
helps you to develop and input a graph ASAP.
In the latter mode you have to make a flawless "String" made specifically for this mode
and use it as input. I made this mode for me and it is super buggy!
To make a "Input string" you basically use every input separated with a space, in order.
If you provide the program with such a string a Bot function will Enter
String Mode Syntax:
`[NodesCount] [theCostOfEdge1] [From] [To] [theCostofEdge2] [From] [To] ...
[-3 (just when you are done with entring inputs)] [StartNode] [DestinationNode]`
e.g. The First Graph in my Documentation about Dijkstra:
    `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`

Extra: Extra things are just Extra (like Updates) and *I* even don't need them
in anycase BUT I thought it's better to keep them here for further development.
(because back then I didn't know such thing as Git exists.)

This program only supports inputs up to 20 nodes.
*/

using System;
using System.Diagnostics.CodeAnalysis;
using System.Text.RegularExpressions;

namespace router
{
    internal class Program
    {
        private const bool  Error           = true,             // When true user receives
        ↪  "Error"s and Feedbacks
                            Reportavailable = true;             // When true you can call
                            ↪  table by -1 in most inputs
```

```
        private const byte  Lettercode      = 64,               // 64 for CAPs 96 for small-caps letter signs
                            Maxnodetosupport= 26;               // as letters are 26 //TODO extend me
        private const int   Infinite        = int.MaxValue;     // Extend Infinite when extend var type : Handwrite*
        public enum Alerts { Error, Info, Notice, Success, Default, DefaultWithBR }
        private static Dijkstra.Node[]  _dijkstraNodes;
        private static int[,]           _nodesTableInts;        // The Graph states and Definition is goes in this table
        private static int              _stringNthRequired = 1; // is used in StringMode and makes string into Fragments
        private static byte             _nodesCount;            // Number of Nodes
        private static string           _string;
        private static char             _sourceNode,
                                        _destinationNode;
        private static bool             _stringMode;

        private static bool Mapper ()
        {
            //Mapper mode to Input the network you want to find shortest path in it (Mapper Mode)

            //Intro
            string header_temp = "Mapper Mode";
            Output.Visuals.HeaderFooter.Start(header_temp);

            //Input: Number of Nodes
            string output_temp = "Enter the number of nodes on your network: ";
            if (!_stringMode)
                _nodesCount = (byte)Input.Int32(output_temp, 3, Maxnodetosupport, checkReportPossible: false);
            else
            {
                _string = Input.String(
                    "(If you don't know what you are doing run DefaultMode by entering 1)" + "\n" +
                    "Input THE string you want to apply:", Alerts.Notice, false);
                _nodesCount = byte.Parse(Job.ExtractNumbers(Input.Bot(output_temp)));
            }


            //Initialize Table of Network States or NodeTable
            _nodesTableInts = new int[_nodesCount + 1, _nodesCount + 1]; //+1 because of Table header and labels
```

```
        //fill the table/array with headers and labels
        _nodesTableInts[0, 0] = -1;
        for (int i = 1; i ≤ _nodesCount; i++)
        {
            _nodesTableInts[0, i] = i + Lettercode;
            _nodesTableInts[i, 0] = i + Lettercode;
            _nodesTableInts[i, i] = -1;
        }


        //Input: Data Entry (for NodeTable)
        Output.Alert("START: Data entry", Alerts.Notice);
        while (true)
        {
            output_temp = "Enter the Cost-Price you want (-3 to break): ";
            int data =
                _stringMode? int.Parse(Job.ExtractNumbers(Input.Bot(output_temp))) :
                ↪ Input.Int32(output_temp, -3);

            if (data < 0)
            {
                if (data == -3)
                {
                    Output.Alert("END: Data entry", Alerts.Notice);
                    break;
                }
                continue;
            }

            char x, y;
            if (_stringMode)
            {
                x = char.Parse(Input.Bot("From: "));
                y = char.Parse(Input.Bot("To: "));
            }
            else
            {
                char endNodeChar = (char)(_nodesCount + Lettercode);
                x = Input.Char("From: ", max: endNodeChar);
                y = Input.Char("To: ", max: endNodeChar);
            }
            if (x ≠ y)
            {
                _nodesTableInts[x - Lettercode, y - Lettercode] = data;
                _nodesTableInts[y - Lettercode, x - Lettercode] = data;
```

```
                continue;
            }
            // ReSharper disable once ConditionIsAlwaysTrueOrFalse
            if (Error) Output.Alert("You cannot move from a place to itself. That's non-sense.", Alerts.Error);
            Output.BR();
        }


        //Outro
        Output.Visuals.HeaderFooter.End(header_temp);
        Output.BR();
        if (_stringMode)
            Output.Alert("There is Your Network! You are all done here.\n" +
                         "You'll go to next session now." + "\n" +
                         "Where you can process this network (print after this message) you made.", Alerts.Success);
        else
            Input.String("There is Your Network! You are all done here.\n" +
                         "Press anykey to go to next part." + "\n" +
                         "Where you can process this network (print after this message) you made.\n" +
                         "Press any key to Continue... ", Alerts.Success);
        Output.BR();
        return true;
    }

    private static bool Router()
    {
        //Router Mode to find the shortest map you gave before in Mapper Mode (Router Mode)
        //Intro
        string header_temp   = "Router Mode";
        Output.Visuals.HeaderFooter.Start(header_temp);

        //Inputs: Start/Source Node and Destination Node
        string output_temp   = "Enter start Node: ";
        if (_stringMode)
        {
            _sourceNode      = char.Parse(Input.Bot( output_temp                ));
            _destinationNode = char.Parse(Input.Bot( "Enter destination node: "));
            if (_sourceNode == _destinationNode)
            {
                Output.Alert(
```

```
                    "FATAL: SOURCE AND DESTINATION CANNOT BE THE SAME. (FATAL ONLY IN
                    ↪  STRING MODE)", Alerts.Error);
                return false;
            }
        }
        else
        {
            char endNodeChar_temp = (char)(_nodesCount + Lettercode);
            _sourceNode           = Input.Char(output_temp, max: endNodeChar_temp);
            while (true)
            {
                _destinationNode  = Input.Char("Enter destination node: ", max:
                ↪  endNodeChar_temp);
                if (_sourceNode  == _destinationNode)
                    Output.Alert("Source and Destination cannot be the same.",
                    ↪  Alerts.Error         );
                else
                    break;
            }
        }

        //Check possibility
        Output.Alert("Checking Connections...", Alerts.Info);
        if (!Job.CheckNodeConnection(_sourceNode))
        {
            Output.Alert("FATAL: START/SOURCE NODE IS DISCONNECTED FROM THE NETWORK!"
            ↪  , Alerts.Error  );
            return false;
        }
        if(!Job.CheckNodeConnection(_destinationNode))
        {
            Output.Alert("FATAL: DESTINATION NODE IS DISCONNECTED FROM THE NETWORK!"
            ↪  , Alerts.Error  );
            return false;
        }
        Output.Alert("Start Node and Destination Node are Connected to the
        ↪  Network...", Alerts.Success);
        Console.WriteLine("Let's see if it's possible to find Shortest Path from Start
        ↪  Node to Destination Node." );

        //Dijkstra Algorithm
        Output.Alert("START: Dijkstra's Process", Alerts.Notice);
        Dijkstra.Run();

        Output.Visuals.HeaderFooter.End(header_temp);
```

```
        return true;
    }
    public static void Main()
    {
        //Intro
        Output.Alert("Usually, when input available," + "\n" +
                     "\t" + "you can get a Table of Network States (NodeTable) by
                     ↪  entering \"-1\".", Alerts.Info  );

        //Input: Mode*
        Output.Alert("How you wanna enter output?"
        ↪  , Alerts.Notice);
        Output.Alert("CAUTION: String Mode (2) Has No Debugger and is NOT stable..." +
        ↪  "\n" +
                     "\t" + "It's generally for test and Develop..." +
                     "\t" + "Do NOT use it if you are not sure about inputs."        ,
                     ↪  Alerts.Error, false        );
        int inputMode_temp =
            Input.Int32("Enter Mapper Mode: (Defualt: '1' or String: '2') ", 1, 2,
            ↪  checkReportPossible: false);
        switch (inputMode_temp)
        {
            case 1:
                _stringMode = false;
                break;
            case 2:
                _stringMode = true;
                break;
        }

        //Mapper mode to Input the network you want to find shortest path in it
        ↪  (Mapper Mode)
        if (!Mapper())
            return;

        Output.Visuals.Table.Int(_nodesTableInts);

        //Router Mode to find the shortest map you gave before in Mapper Mode (Router
        ↪  Mode)
        if (!Router())
            return;

        Console.ReadKey();

    }
```

```
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'
                    ↪  (Temporary) TODO explain more
244                               ForeNode;
245                 public int    PathCost;
246             }
247
248             //We use a -1 when it comes to _dijkstraNodes Array
249             //because array starts at 0 but first array (nodeTable) starts at 1
250
251             private static void Init()
252             {
253                 //Initialize
254                 //Dijkstra: at first we guess all Nodes' PathCost. Source Node = 0 and
                    ↪  rest are Infinite
255                 _dijkstraNodes = new Node[_nodesCount];
256                 for (int i = 0; i < _nodesCount; i++)
257                 {
258                     _dijkstraNodes[i] = new Node
259                     {
260                         //Name = (i + Lettercode + 1).ToString(),
261                         ForeNode = '-',
262                         PathCost = Infinite
263                     };
264                 }
265                 _dijkstraNodes[_sourceNode - Lettercode -1].Status = 'p';
266                 _dijkstraNodes[_sourceNode - Lettercode -1].PathCost = 0;
267             }
268
269
270             private static char[] ReturnNodesToSelectMinBetween(bool
                ↪  FirstTimeFor_sourceNode)
271             {
272                 string nodesToSelectMinBetween = "";
273                 char[] tempNodesConnectedTo    = Get.TempNodesConnectedTo(_sourceNode);
274                 if (FirstTimeFor_sourceNode)
275                     nodesToSelectMinBetween    = Get.StringFromCharArray
                        ↪  (tempNodesConnectedTo);
276                 else
```

```
            for (int i = 0; i < _nodesCount; i++)
            {
                char status;
                try { status = _dijkstraNodes[i].Status; }
                catch { continue; }
                if (status == 't')
                    nodesToSelectMinBetween += (char) (i + Lettercode + 1);
            }
        return Get.CharArrayFromString(nodesToSelectMinBetween);
    }


    private static char SelectMinChar(string nodesToSelectMinBetween, int shift =
    ↪ 0)
    {
        //Dijkstra: Between Candidate nodes (from first step) we select the one
        ↪ with Lowest Cost (minChar = )
        //We Turn the MinChar (Return of this Function) into next CurrentNode
        //Shift doesn't let program to do any loops if all values are same.
        //If there are two candidates for minChar first time it will select first
        ↪ next time it will select next.
        int  c       =  0 ;
        int  min     =  0 ;
        char minChar = '0';
        foreach (char node in nodesToSelectMinBetween)
        {
            if (c == shift)
            {
                min     = _dijkstraNodes[node - Lettercode - 1].PathCost;
                minChar = node;
            }
            else if (c > shift)
            {
                if (_dijkstraNodes[node - Lettercode - 1].PathCost < min)
                {
                    min     = _dijkstraNodes[node - Lettercode - 1].PathCost;
                    minChar = node;
                }
            }
            c++;
        }
        return minChar;
    }
```

```
        private static char[] FindForeNodes(char node, string path_memory = "")
        {
            //This Function Finds the Path from _destinationNode back to the
            ↪ source/start point
            char foreNode = _dijkstraNodes[node - Lettercode -1].ForeNode;
            if (foreNode ≠ '-')
            {
                path_memory += foreNode;
                return FindForeNodes(foreNode, path_memory);
            }
            char[] pathChars = new char[path_memory.Length +1];
            int counter = path_memory.Length -1;
            foreach (char station in path_memory)
                pathChars[counter--] = station;

            pathChars[path_memory.Length] = _destinationNode;
            return pathChars;
        }


        public static void Run()
        {
            Init();
            //Dijkstra: Start from _sourceNode (start point). we set it to
            char   currentNode = _sourceNode;
            Output.Alert("currentNode = " + currentNode, Alerts.Info);
            bool   FirstTimeFor_sourceNode = true;
            string error       = "";
            int    shift       = 0;
            char   minChar     = '-';
            //Dijkstra: Then we select first Node that is connected to _sourceNode
            ↪ (start Point)
            while (true)
            {
                if (error == "fragmental Network") break;
                string nodesToSelectMinBetween =

                    ↪ Get.StringFromCharArray(ReturnNodesToSelectMinBetween(FirstTimeFor_sourceNode
                if (nodesToSelectMinBetween ≠ "")
                     Output.Alert(
                         "Nodes To Select a Minimum Between them are: " +

                          ↪ Output.Arrays.Char(Get.CharArrayFromString(nodesToSelectMinBetween),
                          ↪ false)
                        , Alerts.Info);
```

```
                else
                {
                    Output.Alert
                        ("There is No Node Left To Select a Minimum Between them and
                        ↪  Continue Process! Breaking..."
                        , Alerts.Notice);
                    break;
                }
                char[] tempNodesConnectedTo = Get.TempNodesConnectedTo(currentNode);
                if (tempNodesConnectedTo != null)
                {
                    //skip and redo current node with a new node in
                    ↪  NodesToSelectMinBetween
                    Output.Alert(
                        "Nodes Connected to Node " + currentNode + " are: " +
                         Output.Arrays.Char(tempNodesConnectedTo, false)
                        , Alerts.Info);
                    foreach (char node in tempNodesConnectedTo)
                    {
                        Output.Alert("chosenNode = " + node, Alerts.Info);
                        //Dijkstra: Here We Calculate the PathCost for each node
                        //PathCost = (CurrentNode's PathCost + the Cost of going to
                        ↪  new Node)
                        int pathCost_temp =
                            _dijkstraNodes[currentNode - Lettercode - 1].PathCost +
                            Get.DataFromArray.NodeTableInts(currentNode, node);
                        if (pathCost_temp < 0)
                        {
                            error = "fragmental Network";
                            break;
                        }
                        Output.Alert("totalPathCost = " + pathCost_temp, Alerts.Info);
                        //Dijkstra: If the calculated PathCost is less than previous
                        ↪  PathCost we update it with ...
                            //...new PathCost. Then we set a mark (ForeNode) that
                            ↪  reminds us how we got to this node
                        if (pathCost_temp < _dijkstraNodes[node - Lettercode -
                        ↪  1].PathCost)
                        {
                            Output.Alert("last cost = " + _dijkstraNodes[node -
                            ↪  Lettercode - 1].PathCost,
                                Alerts.Info);
                            _dijkstraNodes[node - Lettercode - 1].PathCost =
                            ↪  pathCost_temp;
```

```
                    _dijkstraNodes[node - Lettercode - 1].ForeNode =
                    ↪  currentNode;
                    Output.Alert("UPDATE: value applied", Alerts.Info, false);
                }
                //Dijkstra: we keep re-doing these steps till we declare state
                ↪  of all nodes which are ...
                    //...connected to the _sourceNode. then we process the
                    ↪  Graph
                //Dijkstra: There are a bit difference in how Dijkstra
                ↪  proceeds the first time (nodes ...
                    //... Connected to the _sourceNode), how it generally runs
                    ↪  in graph and finally how ...
                    //... it does it in last time... PLEASE pay attention to
                    ↪  details of these Method (Run())
            }
        }
        else
        {
            Output.Alert
                ("CAUTION: Failed: Could NOT find any node ahead of this
                ↪  node...", Alerts.Error, false);
            Output.Alert("Retrying other node"
            ↪  , Alerts.Notice      );
        }

        //Dijkstra: Select Minimum Node in this graph to Set as new
        ↪  CurrentNode
        //Shift doesn't let program to do any loops if all values are same.
        //If there are two candidates for minChar first time it will select
        ↪  first, then next...
        char lastMinChar = minChar;
        minChar          = SelectMinChar(nodesToSelectMinBetween, shift  );
        if (!FirstTimeFor_sourceNode &&
            lastMinChar == minChar   &&
            lastMinChar ≠
            ↪  nodesToSelectMinBetween[nodesToSelectMinBetween.Length -1])
        {
            minChar      = SelectMinChar(nodesToSelectMinBetween, ++shift);
        }
        else if (FirstTimeFor_sourceNode)
        {
            minChar      = SelectMinChar(nodesToSelectMinBetween         );
            FirstTimeFor_sourceNode = false;
        }
        else if (lastMinChar == minChar)
```

```
                {
                    break;
                }

                currentNode = minChar;

                Output.Alert("minchar = " + minChar, Alerts.Info);
                //Dijkstra: when here: we restart the whole process to find the path to _destinationNode ...
                    //... (when _destinationNode.Status = Permanent)
                //In my app I process till all nodes go 'p' because that way we have something named 'Shortest ...
                    //... Path Tree' which helps us to find any shortest path from this constant start point. ...
                    //... In other words: My app draws a Shortest Path Tree from Start point to everywhere then ...
                    //... Selects the destination and gets the path via the tree.
            //TODO add a branch which this class is different: so program can draw the tree and print it: no destination!
                if (_dijkstraNodes[currentNode - Lettercode -1].PathCost ≠ Infinite)
                    _dijkstraNodes[currentNode - Lettercode -1].Status = 'p';
                Output.Alert("currentNode's (" + currentNode +") status = " +
                             _dijkstraNodes[currentNode - Lettercode - 1].Status
                            , Alerts.Info);
                Output.BR();
            }
            Output.Alert("END: Dijkstra Process", Alerts.Notice);
            Output.BR(5);


            int pathCost = _dijkstraNodes[_destinationNode - Lettercode - 1].PathCost;
            if (pathCost == Infinite || error == "fragmental Network")
                Output.Alert("This Destination is not Accessable from this Start Node", Alerts.Error);
            else
                Output.Alert("This is the cost of shortest path possible: " +
                             pathCost + " via path between these Nodes: "   +
                             Output.Arrays.Char(FindForeNodes(_destinationNode), false)
                    , Alerts.Success);
        }
    }

    private class Output
    {
```

```
۴۶۷             public static void BR(int number = 1)
۴۶۸             {
۴۶۹                 for (int i = 1; i ≤ number; i++)
۴۷۰                     Console.WriteLine();
۴۷۱             }
۴۷۲             public class Visuals
۴۷۳             {
۴۷۴                 [SuppressMessage("ReSharper", "UnusedMethodReturnValue.Local")]
۴۷۵                 public class HeaderFooter
۴۷۶                 {
۴۷۷                     public static string Start(string data, bool print = true)
۴۷۸                     {
۴۷۹                         data = ("\n╔═══════════════════════════════════════════" +
۴۸۰                                 "\n╠ >>> START: " + data +
۴۸۱                                 "\n╟───────────────────────────────────────────\n");
۴۸۲                         if (print) Alert(data, Alerts.Notice, false);
۴۸۳                         return data;
۴۸۴                     }
۴۸۵                     public static string End(string data, bool print = true)
۴۸۶                     {
۴۸۷                         data = ("\n╟───────────────────────────────────────────" +
۴۸۸                                 "\n╠ >>> END: " + data +
۴۸۹                                 "\n╚═══════════════════════════════════════════\n");
۴۹۰                         if (print) Alert(data, Alerts.Notice, false);
۴۹۱                         return data;
۴۹۲                     }
۴۹۳                 }
۴۹۴                 public class Table
۴۹۵                 {
۴۹۶                     public static void Int(int[,] array)//prog*
۴۹۷                     {
۴۹۸                         string mode_temp = "Int Table Output";
۴۹۹                         HeaderFooter.Start(mode_temp);
۵۰۰                         Console.Write("╟     ");
۵۰۱                         int i;
۵۰۲                         for (i = 1; i < array.GetLength(0); i++)
۵۰۳                             Console.Write("| [" + (char) array[i, 0] + "] ");
۵۰۴                         for (i = 1; i < array.GetLength(0) ; i++)
۵۰۵                         {
۵۰۶                             Console.Write("\n╟ [" + (char) array[i, 0] + "]");
۵۰۷                             for (int j = 1; j < array.GetLength(1); j++)
۵۰۸                                 Console.Write(" | {0, 3}", array[i, j]);
۵۰۹                         }
۵۱۰                         BR();
۵۱۱                         HeaderFooter.End(mode_temp);
```

```
            }
        }
    }
    public class Arrays
        {
            public static string Char(char[] array, bool print = true, bool error
            ↪ = Error)
            {
                if (array == null)
                {
                    if (error) Alert("Null Array to visualize!", Alerts.Error);
                    return "";
                }
                string data = ("{" + array[0]);
                int i = 1;
                if (array.Length == 1)
                    data += ("}");
                else if (array.Length > 1)
                {
                    data += (", ");
                    if (array.Length ≥ 3)
                        for (; i < array.GetLength(0) - 1; i++)
                            data += (array[i] + ", ");
                    data += (array[i] + "}");
                }
                if (print) Console.WriteLine(data);
                return data;
            }
        }
    public static void Alert (string message, Alerts type = Alerts.Default, bool
    ↪ badge = true)
    {
        switch (type)
        {
            case Alerts.Error:
                Console.ForegroundColor = ConsoleColor.DarkRed;
                if (badge) Console.Write("ERROR: ");
                Console.WriteLine(message);
                Console.ForegroundColor = ConsoleColor.Gray;
                break;

            case Alerts.Info:
                Console.ForegroundColor = ConsoleColor.Blue;
                if (badge) Console.Write("INFO: ");
                Console.WriteLine(message);
```

```
                    Console.ForegroundColor = ConsoleColor.Gray;
                    break;

                case Alerts.Notice:
                    Console.ForegroundColor = ConsoleColor.DarkYellow;
                    if (badge) Console.Write("NOTICE: ");
                    Console.WriteLine(message);
                    Console.ForegroundColor = ConsoleColor.Gray;
                    break;

                case Alerts.Success:
                    Console.ForegroundColor = ConsoleColor.Green;
                    if (badge) Console.Write("SUCCESS: ");
                    Console.WriteLine(message);
                    Console.ForegroundColor = ConsoleColor.Gray;
                    break;
                case Alerts.DefaultWithBR:
                    Console.WriteLine(message);
                    break;
                //case Alerts.Default://redundant
                default:
                    Console.Write(message);
                    break;
            }
        }
    }
    private class Input
    {
        public static string String(string message = "Enter a Text String: "
            , Alerts alertType = Alerts.Default
            , bool reports = Reportavailable)
        {
            Output.Alert(message, alertType);
            string value = Console.ReadLine();
            if (!reports) return value;
            switch (value)
            {
                case "-1":
                    Output.Visuals.Table.Int(_nodesTableInts);
                    break;
            }
            return value;
        }//Handwrite/Hardcore*
        public static string Bot(string message = "", bool printValue = true, bool
        ↪  addNth = true)
```

```
        {
            Console.Write(message);
            string value = Get.StringsNthpart(_stringNthRequired);
            if (addNth) _stringNthRequired++;
            if (printValue) Output.Alert("BOT INPUT: " + value, Alerts.Notice);
            Output.BR();
            return value;
        }
        public static int Int32(string message = "Enter an Integer: "
            , int min = int.MinValue
            , int max = int.MaxValue
            , bool error = Error
            , bool checkReportPossible = Reportavailable)
        {
            if (max < min)
            {
                int _temp = min;
                min = max;
                max = _temp;
            }
            int value;
            do
            {
                Console.Write(message);
                string inputString = String("",reports: checkReportPossible);
                if (!string.IsNullOrWhiteSpace(inputString))
                {
                    bool negativeFlag = inputString.Contains("-");
                    inputString = Regex.Match(inputString, @"\d+").Value;
                    if (!string.IsNullOrEmpty(inputString))
                    {
                        value = int.Parse(inputString);
                        if (negativeFlag) value = -value;
                        if (min > value && error)
                            Output.Alert("Your input is so small. Minimum is " + min +
                            ↪  ".", Alerts.Error);
                        else
                        if (max < value && error)
                            Output.Alert("Your input is so big. Maximum is " + max +
                            ↪  ".", Alerts.Error);
                        else break;
                    }
                    if (error) Output.Alert("Invalid input", Alerts.Error);
                }
                else if (error) Output.Alert("Empty input", Alerts.Error);
```

```
            } while (true);
            return value;
        }
        public static char Char(string message = "Enter a Character: "
            , char min = 'A'
            , char max = 'Z'
            , bool error = Error
            , bool checkReportPossible = Reportavailable)
        {
            char value;
            do
            {
                Console.Write(message);
                string inputString = String("", reports: checkReportPossible);
                if (!string.IsNullOrWhiteSpace(inputString))
                {
                    if (inputString.Length > 1)
                    {
                        if (error) Output.Alert("Only 1 character input",
                        ↪  Alerts.Error);
                    }
                    else
                    {
                        value = char.Parse(inputString.ToUpper());
                        if (min > value && error)
                            Output.Alert("Your input is irregular. start is " + min +
                            ↪  ".", Alerts.Error);
                        else if (max < value && error)
                            Output.Alert("Your input is irregular. end is " + max +
                            ↪  ".", Alerts.Error);
                        else break;
                    }
                }
                else if (error) Output.Alert("Empty input", Alerts.Error);
            } while (true);
            return value;
        }
    }
    private class Get
    {
        public class DataFromArray
        {
            public static int NodeTableInts(char x, char y)//prog
            {
                y = char.Parse(y.ToString().ToUpper());
```

```
            x = char.Parse(x.ToString().ToUpper());
            return _nodesTableInts[x - Lettercode, y - Lettercode];
        }
    }

    public static string StringsNthpart(int nth) //dev Mode
    {
        string value = "";
        int part = 1;
        foreach (char cha in _string)
        {
            if (cha != ' ')
                if (part == nth)
                    value += cha;
            else
                part++;
        }
        return value.ToUpper();
    }

    [SuppressMessage("ReSharper", "UnusedMember.Local")]
    public static int ArrayDimensionsCount(Array arrayName) //Extra*
    {
        int dimensionCount;
        for (int i = 0;; i++)
            try
            {
                // ReSharper disable once ReturnValueOfPureMethodIsNotUsed
                arrayName.GetLength(i);
            }
            catch
            {
                dimensionCount = i - 1;
                break;
            }
        return dimensionCount;
    }

    [SuppressMessage("ReSharper", "UnusedMember.Local")]
    public static int NumberOfLetterInString(string data, char letter) //Extra
    {
        int c = 0;
        foreach (char ch in data)
            if (ch == letter) c++;
        return c;
```

```
        }

        [SuppressMessage("ReSharper", "UnusedMember.Local")]
        public static bool UniqueCharsState(string data) //Extra
        {
            if(data.Length > 256)
                return false;
            bool[] char_set = new bool[256];
            foreach (int val in data)
            {
                if(char_set[val])
                    return false;
                char_set[val] = true;
            }
            return true;
        }

        public static char[] TempNodesConnectedTo(char thisNode)
        {
            string data = "";
            for (int i = 1; i ≤ _nodesCount; i++)
            {
                int nodeTableReturn = DataFromArray.NodeTableInts(thisNode, (char)(i +
                ↪ Lettercode));

                if (nodeTableReturn > 0 && _dijkstraNodes[i -1].Status == 't')
                    data += (char)(i + Lettercode);
            }

            if (data.Length == 0)
                return null;
            char[] nodesConnectedToSource = new char[data.Length];
            int c = 0;
            foreach (char cha in data)
            {
                nodesConnectedToSource[c] = cha;
                c++;
            }
            return nodesConnectedToSource;
        }

        public static char[] CharArrayFromString(string data)
        {
            if (data.Length == 0)
                return null;
```

```
            char[] CharData = new char[data.Length];
            int c = 0;
            foreach (char cha in data)
            {
                CharData[c] = cha;
                c++;
            }
            return CharData;
        }

        public static string StringFromCharArray(char[] array)
        {
            string value = "";
            if (array != null)
                foreach (char cha in array)
                    value += cha;
            return value;
        }
    }
    private class Job
    {
        [SuppressMessage("ReSharper", "UnusedMember.Local")]
        private static string ConvertIntToAnyString(int value, int cap) //Extra / For
        ↪ Update*
        {
            string result = "";
            char[] baseChars = new char[cap];
            for (int i = 0; i < cap; i++)
                baseChars[i] = (char)(Lettercode + i);
            int targetBase   = baseChars.Length;
            do
            {
                char letter = baseChars[value % targetBase];
                if (letter == '@') letter = '0';
                result = letter + result;
                value  = value / targetBase;
            }
            while (value > 0);
            return result;
        }

        public static string ExtractNumbers(string data)
        {
            int value = 0;
            if (!string.IsNullOrWhiteSpace(data))
```

```
                {
                    bool negativeFlag = data.Contains("-");
                    data = Regex.Match(data, @"\d+").Value;
                    if (string.IsNullOrEmpty(data)) return value.ToString();
                    value = int.Parse(data);
                    if (negativeFlag) value = -value;
                }
                else return null;
                return value.ToString();
            }

            public static bool CheckNodeConnection(char node)
            {
                for(int i = 1; i ≤ _nodesCount; i++)
                    if (Get.DataFromArray.NodeTableInts(node, (char) (i + Lettercode)) >
                    ↪ 0)
                        return true;
                return false;
            }
        }
    }
}
```