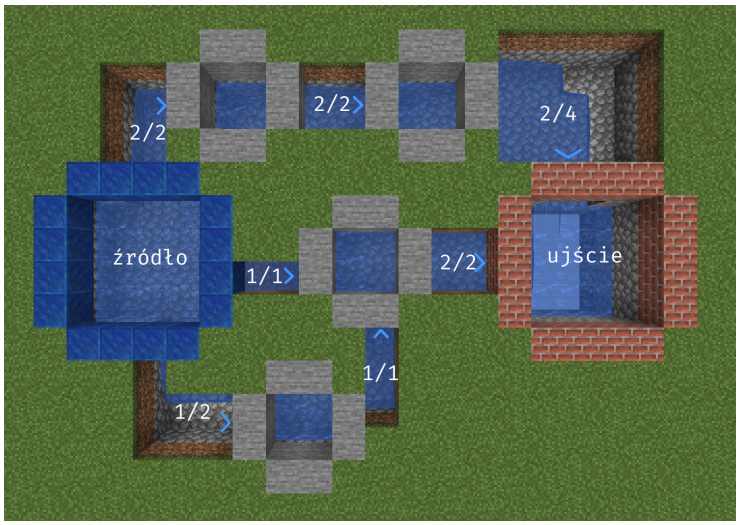
Ezoteryczne Kartki
Przepływy

Jakub Bachurski

wersja 1.1.5

1 Czym jest przepływ?

Zabawę zaczniemy od intuicyjnego i formalnego powiedzenia, czym właściwie są te przepływy. Intuicyjnie, mówiąc o przepływach myślimy o pewnym grafie skierowanym, który modeluje rury (krawędzie) łączące różne zbiorniki (wierzchołki). Wyróżniamy też dwa specjalne zbiorniki: źródło i ujście. W źródle znajduje się nieskończone źródełko wody, którą za pomocą rur (mają one ograniczoną wielkość, więc jedną rurą nie możemy przenosić dowolnie dużo wody) chcemy przetransportować do ujścia, gdzie woda jest zużywana i znika. W zbiornikach nie możemy magazynować wody, pełnią one jedynie rolę skrzyżowania rur.

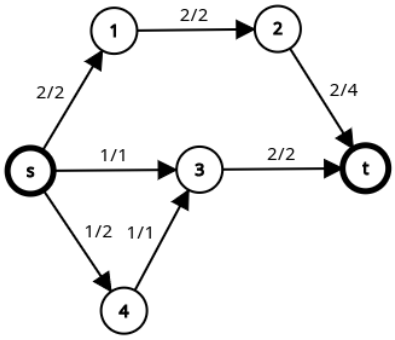


Formalnie, rozważamy *sieć przepływową*, czyli graf skierowany $G = (V, E)$, taki, że dla każdej krawędzi $e \in E$ mamy zdefiniowaną *przepustowość* oznaczaną przez $c(e)$ *(capacity)*, która jest nieujemną liczbą rzeczywistą. Do tego wyróżniamy też wierzchołki: źródło s *(source)* oraz ujście t *(target)*.

Przepływem nazywamy przyporządkowanie każdej krawędzi wartości przechodzącego przez nią przepływu. Przepływ f musi:

- Nie przekraczać przepustowości żadnej krawędzi e: $0 \leqslant f(e) \leqslant c(e)$.
- Spełniać zachowanie przepływu dla każdego wierzchołka $v \neq s, t$ (przepływ wchodzący jest równy przepływowi wychodzącemu):

$$\sum_{u \xrightarrow{e} v} f(e) = \sum_{v \xrightarrow{e} u} f(e)$$



Powyższy obrazek jest bardzo częstą metodą przedstawiania sieci przepływowych: podpis krawędzi f/c oznacza przepustowość c i obecny przepływ f w kierunku wskazanym przez strzałkę.

Logicznym krokiem jest teraz zoptymalizować przepływ w takiej sieci, i warto móc jakoś zmierzyć „optymalność" przepływu. W tym celu definiujemy *wartość przepływu*: przepływ wychodzący ze źródła (równy przepływowi wchodzącemu do ujścia), przy czym musimy pamiętać o tym, że przepływ może wrócić do źródła (wyjść z ujścia):

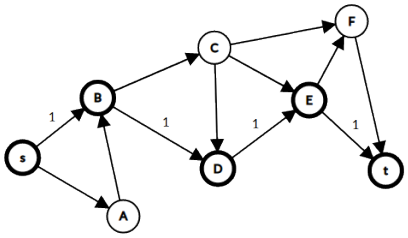
$$F = \sum_{s \xrightarrow{e} u} f(e) - \sum_{u \xrightarrow{e} s} f(e) = \sum_{u \xrightarrow{e} t} f(e) - \sum_{t \xrightarrow{e} u} f(e)$$

Oczywiście, zwykle chcemy znaleźć najbardziej optymalne rozwiązanie. Zatem w naturalny sposób naszym oczom ukazuje się **problem maksymalnego przepływu**, polegający na znalezieniu przepływu o największej możliwej wartości. W przykładowej sieci przepływowej pokazany przepływ jest maksymalny, bo nie istnieje przepływ o większej wartości.

2 Jak znaleźć maksymalny przepływ?

Praktykując algorytmikę, przede wszystkim interesuje nas znajdowanie rzeczy, więc nadszedł czas na próbę stworzenia algorytmu znajdującego maksymalny przepływ. W tym celu rozważymy klasyczną **metodę Forda-Fulkersona**.

Spróbujmy zrobić krok w stronę rozwiązania, zapisując jak mogą wyglądać operacje, które zwiększają wartość przepływu w grafie. Weźmy dowolny maksymalny przepływ i skupmy się na pewnej jednostce przepływu wychodzącej ze źródła. Zauważmy, że po tym, jak dojdzie ona do następnego wierzchołka, musi się gdzieś podziać (bo bilans – różnica przepływu wejściowego i wyjściowego – tego wierzchołka jest teraz dodatni), więc ponownie „przepłynie" w stronę następnego wierzchołka. I tak dopóki nie trafi do ujścia, które nie musi spełniać zachowania przepływu. Intuicja może nas doprowadzić do stwierdzenia, że każdy przepływ (w szczególności maksymalny) może zostać rozłożony na ścieżki, z których każda przenosi jednostkę przepływu.



Metoda Forda-Fulkersona opiera się na inkrementalnym poprawianiu przepływu. W skrócie, działa na bardzo prostej zasadzie: znajduje ścieżkę (którą nazywamy *ścieżką powiększającą*), na której możemy „przepchnąć" więcej przepływu, aktualizuje przepływ na tej ścieżce oraz wartość maksymalnego przepływu, a gdy taka ścieżka nie istnieje, przerywa działanie i stwierdza, że znalazła maksymalny przepływ. Na mocy powyższej obserwacji istnieje taki wybór ścieżek, że w ten sposób znajdziemy maksymalny przepływ.

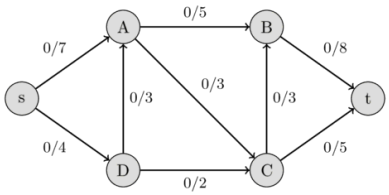
Przejdźmy do szczegółów. Weźmy naszą sieć przepływową i zacznijmy od zerowego przepływu ($f(e) = 0$ dla każdej krawędzi e). Wybierzmy też ciąg krawędzi $(e_1, e_2, e_3, ..., e_n)$, tworzący ścieżkę z s do t. Kluczowy fakt: jeżeli dla każdej z krawędzi na ścieżce zwiększymy przepływ o Δf, to wartość przepływu zwiększy się o Δf, a przepływ dalej będzie poprawny – o ile nie przekroczymy żadnej z przepustowości. Dokładniej, nowy przepływ będzie poprawny, jeżeli

$$\Delta f \leqslant \min_i \{c(e_i) - f(e_i)\}$$

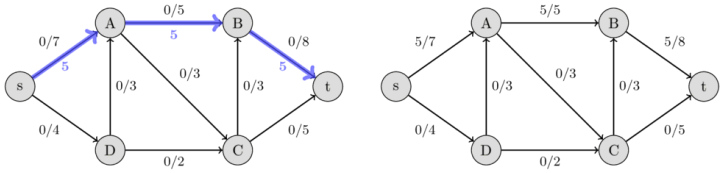
Nietrudno się domyślić, że taka ścieżka jest *powiększająca*, o ile możemy wybrać $\Delta f > 0$. (Jednak za chwilę okaże się, że czegoś w tej definicji brakuje, aby w całości pokrywała się ona z Fordem-Fulkersonem). Ten proces wybierania ścieżek i powiększania przepływu możemy powtarzać wielokrotnie.

2.1 Działanie metody Forda-Fulkersona

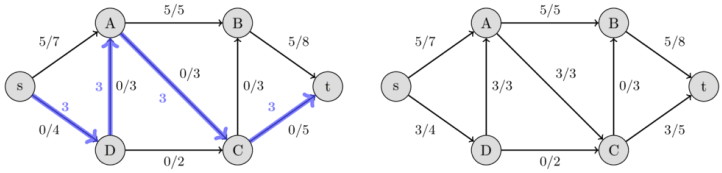
Spróbujmy rozważyć działanie strategii, w której wybieramy dowolną ścieżkę powiększającą, dopóki takowa istnieje. Zacznijmy od takiego prostego grafu[1]:



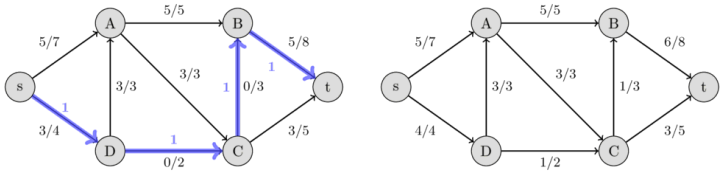
Zacznijmy od wybrania ścieżki powiększającej $s \to A \to B \to t$, która pozwala zwiększyć wartość przepływu o 5 – zatem od razu zwiększamy przepływ na każdej z nich o 5.



Bez większych problemów możemy teraz wybrać $s \to D \to A \to C \to t$, co z kolei zwiększa przepływ o 3 (jesteśmy ograniczeni przez przepustowość $A \to C$).

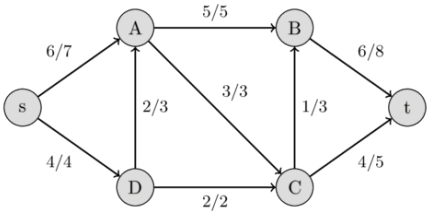


Możemy teraz jeszcze dopchać przez krawędź $s \to D$ jeszcze 1 jednostkę przepływu, przechodząc $s \to D \to C \to B \to t$.



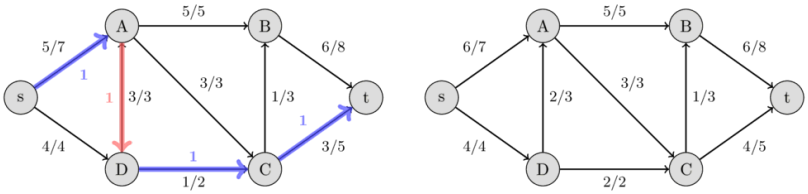
Na obrazku nie widać już żadnej ścieżki powiększającej, zatem metoda w obecnej postaci jest **błędna**, ponieważ istnieje (nie jedyny) następujący przepływ o większej wartości (10 zamiast 9):

[1] Te obrazki oraz cały przykład pochodzą z artykułu na `cp-algorithms.com`.



Przyjrzyjmy się naszemu błędowi: przepchnęliśmy z D do A zbyt dużo, przez co nie wykorzystaliśmy kompletnie pozostałej przepustowości krawędzi $D \to C$ i zapchaliśmy krawędzie wychodzące z A. Metoda ma problem: po zwiększeniu przepływu krawędzi, później nigdy nie jest w stanie go zmniejszyć.

Przychodzi na myśl prosty pomysł: zamiast wybierać krawędzie w mądrzejszy sposób, pozwólmy na *naprawianie błędów* poprzez przepychanie przepływu w drugą stronę. Dokładniej, możemy przepchnąć 1 jednostkę z s do A (bilans wejścia/wyjścia A to $+1$), następnie „odepchnąć" 1 z A do D (krawędź to $D \to A$ – teraz bilans A jest w porządku, ale bilans D to $+1$), a dalej zwyczajnie idziemy przez $D \to C \to t$. Po takiej operacji spełnione jest zachowanie przepływu.



Okazuje się, że tak opisana procedura (poszukiwanie ścieżki powiększającej, dopóki istnieje, przy czym pozwalamy na wycofywanie przepływu) jest w pełni poprawna i właśnie ona nosi nazwę **metody Forda-Fulkersona**.

2.2 Notacja, terminologia i szczegóły

Poszukując krawędzi wchodzących w skład ścieżki powiększającej, niezbyt obchodzą nas krawędzie, dla których $c(e) = f(e)$, bo nie da się przez nie przepchnąć więcej przepływu. Ponieważ w ten sposób się wyróżniają, nazywamy je krawędziami *nasyconymi*. Samej ścieżki powiększającej szukamy w lekko zmodyfikowanej sieci, nazywanej *siecią rezydualną* (oznaczmy ją G_f), która nie zawiera krawędzi nasyconych: pozostałe krawędzie też mają zdefiniowaną przepustowość oznaczaną c_f:

$$c_f(e) = c(e) - f(e)$$

Do tego dla każdej krawędzi e dodaje się w sieci rezydualnej przeciwną krawędź e', dla której zachodzi

$$c_f(e') = f(e)$$

Dzięki takiej definicji, przepustowość w sieci rezydualnej mówi nam, ile jeszcze możemy przepchnąć przepływu przez daną krawędź (dla stowarzyszonej: odepchnąć przez oryginał), a do tego ścieżka powiększająca to każda ścieżka między s a t w tej sieci. Δf jest ograniczone od góry przez minimum po przepustowościach rezydualnych na tej ścieżce.

Zapiszmy jeszcze metodę Forda-Fulkersona pseudokodem:

- Ford-Fulkerson. Wejście: Sieć G, s i t. Zwraca maksymalny przepływ.
- Skonstruuj sieć rezydualną G_f z G. Weź zerowy przepływ f.
- Dopóki w G_f istnieje dowolna ścieżka powiększająca (ścieżka z s do t):
  - Znajdź minimalną przepustowość na pewnej ścieżce P między s i t w G_f i oznacz ją Δf.
  - Przepchnij Δf krawędziami P. Zaktualizuj G_f i f.
- Zwróć f.

Osobiście w implementacji utrzymuję sieć rezydualną. Graf jest przechowywany jako lista sąsiedztwa, w której dla każdego wierzchołka są zapisane wychodzące krawędzie (każda krawędź e umie szybko odnaleźć swojego towarzysza e'). Po znalezieniu w sieci rezydualnej ścieżki powiększającej, wystarczy zaktualizować przepustowości, pamiętając o towarzyszach. Na koniec wystarczy przejrzeć tylko krawędzie, które znajdowały się w oryginalnym grafie, a wynikowy przepływ dla krawędzi e to końcowe $c_f(e')$.

Przerwa na zadanie
„Sieć"

Treść W pewnej sieci komputerowej chcemy wysłać k dużych plików między komputerami A oraz B. Komputery są połączone kablami i dla uproszczenia przyjmijmy, że te połączenia są jednokierunkowe. Stwierdź, czy jest możliwy taki przesył plików, że każde połączenie kablem jest wykorzystane co najwyżej raz.

Rozwiązanie Przypomnijmy sobie, że ścieżka między parą wierzchołków w grafie może odpowiadać przepływowi jednostki przepływu. Rozważmy więc naszą sieć jako graf, a następnie jako sieć przepływową, ze źródłem A oraz ujściem B. Nadajmy też każdej krawędzi przepustowość 1. Jeżeli w tym grafie istnieje przepływ wielkości k (a zatem maksymalny przepływ $F \geqslant k$) to rozwiązanie istnieje i możemy je odtworzyć prostym DFSem.

Problem znalezienia k rozłącznych ścieżek to tylko wierzchołek góry lodowej, jeżeli chodzi o problemy optymalizacyjne rozwiązywane maksymalnym przepływem.

2.3 Przekroje. Twierdzenie o maksymalnym przepływie i minimalnym przekroju

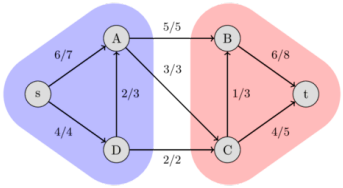
Przechodzimy teraz do kolejnego bardzo przydatnego zagadnienia, wbrew pozorom bardzo blisko związanego z przepływami. Z jego pomocą będziemy też w stanie udowodnić poprawność metody Forda-Fulkersona.

Klasyczne twierdzenie Max-Flow Min-Cut (flow – przepływ, cut – przekrój) to jedno z najbardziej zaskakujących i przydatnych fragmentów teorii związanej z przepływami. Rozważanie go zaczniemy od zdefiniowania samych przekrojów.

Weźmy ważony graf skierowany $G = (V, E)$ z wyróżnionymi dwoma wierzchołkami a oraz b. Stwórzmy zbiór wierzchołków A oraz zbiór pozostałych wierzchołków $B = V \setminus A$, tak, że $a \in A$ i $b \in B$. Parę (A, B) nazywamy *a-b przekrojem*. Nazwa bierze się z procesu „odcinania" wierzchołków A od B, tak, żeby nie dało się przejść z tych pierwszych do drugich. Aby to zrobić, należy usunąć (przeciąć) wszystkie krawędzie prowadzące z grupy A do grupy B. Za pomocą tego procesu definiujemy *wartość przekroju*: jest to suma wag przeciętych krawędzi. *Minimalny a-b przekrój* to taki, że jego wartość jest najmniejsza spośród wszystkich a-b przekrojów tego grafu. Warto zauważyć, że minimalny przekrój stanowi też najtańszy sposób usunięcia krawędzi tak, że nie ma ścieżki $a \rightsquigarrow b$.

Zanim przejdziemy do twierdzenia, warto jakoś powiązać ze sobą pojęcia przepływu i przekroju. Zauważmy, że każdy maksymalny przepływ w sieci ze źródłem s i ujściem t, w pewien sposób generuje nam s-t przekrój. Dokładniej, przyjrzyjmy się sieci rezydualnej dla pewnego maksymalnego przepływu: jeżeli jest on maksymalny, to z pewnością nie może być w nim ścieżki powiększającej.

Utwórzmy przekrój, w którym A to wierzchołki osiągalne ze źródła s w sieci rezydualnej (B to pozostałe) – potraktujmy przy tym przepustowości krawędzi jako wagi. Wtedy przepływ, który opuszcza A jest od góry ograniczony przez wartość przekroju (A, B): w istocie, to ograniczenie to suma przepustowości krawędzi, które wychodzą z A do B.



Nie pozostaje nam nic jak wprowadzić twierdzenie o maksymalnym przepływie i minimalnym przekroju:

Twierdzenie (Max-Flow Min-Cut)**.** *Wartość maksymalnego przepływu jest równa wartości minimalnego s-t przekroju.*

Z pomocą tego twierdzenia możemy udowodnić poprawność metody Forda-Fulkersona. Chętnych poznania dowodów wszystkich twierdzeń odsyłam do prezentacji z uniwersytetu Princeton, gdzie znajdują się ich szkice:

`https://www.cs.princeton.edu/~wayne/kleinberg-tardos/pdf/07NetworkFlowI.pdf`

Przerwa na zadanie
„Sysadmin"

Treść Jesteś sysadminem, i masz dosyć wymagań swoich nieobeznanych z informatyczną sztuką pracodawców. Z związku z tym, w firmowej sieci (której komputery są połączone jednokierunkowo) zamierzasz tak przeciąć kable (tudzież przerwać połączenia), aby nie było pośredniego ani bezpośredniego połączenia między głównymi serwerami: A oraz B. Chcesz się dowiedzieć, czy da się to zrobić przecinając k kabli.

Rozwiązanie To zadanie nieprzypadkowo przypomina poprzednie. Teraz jest to oczywiste, że możemy modelować sieć połączeń jako sieć przepływową, w której chcemy znaleźć minimalny A-B przekrój. Jak już wiadomo, można go łatwo znaleźć za pomocą maksymalnego przepływu – wystarczy sprawdzić, czy jest on mniejszy bądź równy k.

Zauważmy, że graf połączeń konstruujemy w dokładnie ten sam sposób co ostatnio, i zarówno pokrycie ścieżkami rozłącznymi krawędziowo, jak i przecinane kable liczymy tak samo. Przed nami pojawia się **Twierdzenie Mengera**, mówiące, że graf ma taką samą minimalną wielkość A-B przekroju, ile maksymalnie da się wybrać rozłącznych ścieżek między A i B. Dosyć łatwo zauważyć, że Max-Flow Min-Cut jest generalizacją tego twierdzenia.

2.4 Algorytmy

Metoda Forda-Fulkersona nieprzypadkowo nie jest nazywana algorytmem: w jej opisie nie jest uwzględniony sposób szukania ścieżek powiększających. Od użytej strategii znacznie zależy wynikowa złożoność. Wybierając dowolne ścieżki powiększające w sieci o całkowitych przepustowościach, pesymistycznie każda z nich zwiększy wartość przepływu tylko o 1. Wtedy otrzymamy pesymistyczną złożoność $O(FE)$ (częściej jest to zapisywane jako $O(f \cdot E)$). W ramach ciekawostki: okazuje się, że istnieją takie przypadki, że niepoprawny wybór ścieżek powiększających prowadzi do sytuacji, w której algorytm nigdy się nie zakończy. Korzystają one z niewymiernych przepustowości na krawędziach i nie są problemem dla poniższych algorytmów.

Warto zaznaczyć, że w praktyce algorytmy przepływowe działają dużo szybciej, niż zakłada ich złożoność (oczywiście bez przesady, liniowe to one nie są).

2.4.1 Edmonds-Karp

W bardzo prosty sposób możemy polepszyć złożoność Forda-Fulkersona, znajdując ścieżki w sieci rezydualnej za pomocą przeszukiwania wszerz. Dzięki temu do znalezienia maksymalnego przepływu potrzebne będzie co najwyżej $O(VE)$ ścieżek, każda w złożoności $O(V+E)$, co daje złożoność rzędu $O(VE^2)$. Jest to klasyczna metoda szukania przepływów w programowaniu konkursowym. Nosi nazwę **algorytmu Edmondsa-Karpa**. Jego złożoność pokazuje, że na pierwszy rzut oka nieznaczne heurystyki mogą mieć duże znaczenie.

2.4.2 Dinic

Lepszą metodą szukania maksymalnego przepływu jest **algorytm Dinica** [dinika] (nazywany też czasem algorytmem **Dinitza** [dinica], ze względu na inny zapis nazwiska autora: czasem rozróżnia się też zapisem nazwy ulepszoną wersję oryginalnego algorytmu). Dzięki modyfikacji procesu szukania ścieżek powiększających osiąga on zauważalnie lepszą złożoność $O(V^2E)$. Idea Dinica jest prosta: Edmonds-Karp pokazuje nam, że opłaca się szukać najkrótszych ścieżek powiększających. Spróbujmy zatem za jednym razem rozpatrywać wszystkie ścieżki powiększające danej długości, zaczynając od najkrótszych.

Działa on w następujący sposób: przed poszukiwaniem ścieżek powiększających, konstruowany jest *graf poziomowy* G_L sieci rezydualnej. Jest to graf, w którym pozostawione są jedynie krawędzie leżące na jakiejś najkrótszej ścieżce z s do t. Możemy go łatwo stworzyć, licząc odległości do wszystkich wierzchołków z s (odległość do v to d_v) i pozostawiając jedynie krawędzie $u \to v$, dla których $d_v = d_u + 1$ – jest on w oczywisty sposób acykliczny. Następnie, rozważając jedynie krawędzie grafu poziomowego, znajdujemy największy przepływ (nazywany *przepływem blokującym* – nie jest to globalny maksymalny przepływ, a jedynie taki, który możemy uzyskać, korzystając tylko z tych krawędzi). Możemy to osiągnąć, wielokrotnie używając DFS-a w poszukiwaniu ścieżki powiększającej (BFS nie ma tutaj sensu, bo w G_L wszystkie ścieżki z s do t są tej samej długości). Po znalezieniu przepływu blokującego, od nowa liczymy G_L. Aby usprawnić poszukiwanie przepływu blokującego, podczas przeszukiwań wgłąb dla każdego wierzchołka pamiętamy, przez które z krawędzi nie da się przesłać więcej przepływu (jeżeli raz otrzymamy informację, że nie da się tamtędy nic przesłać, to już nigdy nie sprawdzamy tej krawędzi).

Można pokazać, że długość najkrótszej ścieżki powiększającej musi się zwiększać, zatem faz algorytmu będzie $O(V)$. Przepływ blokujący liczymy w $O(VE)$, a sam graf poziomowy w $O(V+E)$ jednym BFS. Daje nam to obiecywaną złożoność $O(V^2E)$.

`https://cp-algorithms.com/graph/dinic.html`

Ciekawostką jest usprawnienie Dinica, działające w pesymistycznej złożoności $O(VE \log C)$ dla grafów z całkowitymi przepustowościami ograniczonymi przez C. Korzysta z idei *algorytmów* skalujących – jest to tak zwany Dinic ze skalowaniem. Pomysł jest bardzo prosty: zacznijmy modyfikowanie przepływu

od krawędzi o dużej przepustowości, z czasem rozpatrując krawędzie o coraz mniejszej przepustowości. Dokładniej, wybieramy pewną wartość k, a w trakcie liczenia grafu poziomowego bierzemy pod uwagę jedynie krawędzie, które w tym momencie miały przepustowość rezydualną $c_f(e) \geqslant 2^k$ (na początku $k = \max\lfloor \log_2 c_f(e) \rfloor$). Gdy skończymy (nie istnieje ścieżka z s do t w grafie poziomowym dla tego k), zmniejszamy k o 1. W praktyce, może to zauważalnie zwiększyć czynnik stały, ale z drugiej strony zmniejsza pesymistyczną złożoność. W zadaniach, gdzie pojawiają się pesymistyczne przypadki, zastosowanie Dinica ze skalowaniem może być kluczowe.

2.4.3 Inne algorytmy

Innym algorytmem o dobrej złożoności jest *Push-Relabel*, działający w $O(VE^2)$, a z pewnym usprawnieniem w $O(VE + V^2\sqrt{E}) = O(V^3)$.

`https://cp-algorithms.com/graph/push-relabel.html`
`https://cp-algorithms.com/graph/push-relabel-faster.html`

Warto też wspomnieć o algorytmie trzech Hindusów (MPM), osiągającym $O(V^3)$:

`https://cp-algorithms.com/graph/mpm.html`

Ciekawym nowym algorytmem jest *Tidal Flow*, zaprezentowany na konferencji IOI w 2018 roku. Czasem osiąga on wyniki lepsze niż Dinic, lecz jego udowodniona złożoność to $O(VE^2)$ i istnieją testy, na których ma trochę gorszą wydajność. Można o nim przeczytać w oryginalnej pracy:

`https://ioinformatics.org/journal/v12_2018_25_41.pdf`

Tak jak wcześniej wspomniałem, w programowaniu konkursowym nie warto kombinować, bo rzadko pojawiają się testy powodujące pesymistyczne zachowanie powyższych algorytmów. Najlepiej pisać algorytm, który najlepiej się umie, a jest wystarczający w danym zadaniu.

3 Po co komu przepływy?

The spice must flow.

Problem maksymalnego przepływu opisano już w 1954 roku, a rozwiązanie wielomianowe opublikowane zostało w 1955 przez Forda i Fulkersona (jest to właśnie opisana wyżej metoda). Możemy wykorzystać go w różnorodnych problemach, korzystając przy tym także z Max-Flow Min-Cut. Zarówno maksymalny przepływ, jak i minimalny przekrój znajdują wiele zastosowań w wielu problemach optymalizacyjnych:

- Zarządzanie: przydzielanie lotów ekipom pokładowym linii lotniczej.
- Procesowanie grafiki: segmentacja obrazu na tło oraz pierwszy plan, mając dane pewne heurystyki.
- Skojarzenia w grafie dwudzielnym: problem łączenia obiektów w pary może być rozwiązany przepływami i znajduje wiele zastosowań.
- Problem wyboru projektu: różne projekty dają zysk, ale mogą wymagać wykonania innych projektów, których wykonanie może powodować straty.

4 Modyfikacje przepływów

Zanim przejdziemy do wyczekiwanych zadanek, warto przejść przez parę modyfikacji przepływów, które będą miały dla nas duże znaczenie przy wymyślaniu rozwiązania. Powiemy też o niespodziewanym miejscu, w którym pojawiają się przepływy.

4.1 Drobne modyfikacje Max Flow

Gdy wprowadzona do problemu maksymalnego przepływu zmiana jest nieznaczna, zwykle możemy sobie poradzić modyfikując sieć przepływową.

4.1.1 Graf nieskierowany

Warto zacząć rozważania od nieskierowanych krawędzi w sieci przepływowej. Mianowicie, dowolną krawędź $u \leftrightarrow v$ wystarczy zastąpić krawędziami $u \to v$ oraz $v \to u$ o takiej samej przepustowości. Taki zabieg można łatwo uzasadnić: gdy będziemy szukali maksymalnego przepływu, z pewnością istnieje takie rozwiązanie, że przepływ płynie tylko w jedną ze stron (albo wcale). Dowód pozostawiony jako ćwiczenie dla czytelnika `:)`.

4.1.2 Przepustowość wierzchołków

Innym pojawiającym się utrudnieniem w zadaniu może być obecność przepustowości na wierzchołku – to znaczy, że ograniczony jest przechodzący przez niego przepływ (suma przepływu wychodzącego). Aby sobie z tym poradzić, chcemy stworzyć krawędź, która będzie symbolizowała przejście przepływu przez wierzchołek. Zatem podzielmy każdy wierzchołek v na wierzchołki v_{in} oraz v_{out}, i między nimi wstawmy krawędź $v_{in} \to v_{out}$ o przepustowości q. Teraz wystarczy zastąpić krawędzie $u \to v$ przez $u_{out} \to v_{in}$, a $v \to u$ przez $v_{out} \to u_{in}$.

4.1.3 Wiele źródeł i ujść

W wypadku, gdy mamy do czynienia z wieloma źródłami $s_1, s_2, ..., s_k$ i/lub wieloma ujściami $t_1, t_2, ..., t_l$, wystarczy stworzyć superźródło s i superujście t, i połączyć je krawędziami o przepustowości ∞ z istniejącymi: $s \to s_i$, $t_i \to t$.

4.1.4 Przepływy z żądaniami

Kolejnym pomysłem jest generalizacja nierówności przepustowości, poprzez dodanie dolnego ograniczenia na przepływ: *żądanie* $d(e)$, takie że $d(e) \leqslant f(e) \leqslant c(e)$. Wtedy zerowy przepływ niekoniecznie musi spełniać nowe wymagania, więc powstaje problem decyzyjny: czy istnieje przepływ spełniający nierówności i zachowujący warunek zachowania przepływu. Jego rozwiązanie jest bardziej złożone i można o nim przeczytać `[tutaj]`. Ten problem jest związany z problem krążenia (cyrkulacji), w którym nie są stricte wyróżnione źródło i ujście.

4.2 Przepływ o minimalnym koszcie

Wstęp Bardzo przydatnym rozszerzeniem przepływów jest przepływ o minimalnym koszcie. Dokładniej, dla każdej krawędzi wprowadzamy koszt $k(e)$, taki, że kosztem przepływu nazywamy sumę $\sum_e f(e) \cdot k(e)$. Sam problem można sformułować teraz na kilka sposobów: na przykład, może być to znalezienie maksymalnego przepływu, którego koszt nie przekracza K (można tutaj też narzucić, aby koszt był liczbą całkowitą), albo żeby spośród maksymalnych przepływów wybrać dowolny o minimalnym koszcie. Rozwiązanie obu z tych problemów jest bardzo podobne.

Idea Pomysł opiera się na „zbieraniu" ścieżek powiększających, które pozwalają nam jak najtaniej przepchnąć dowolną ilość przepływu. Idzie za tym intuicja, że z czasem będziemy otrzymywali rosnące koszty takich ścieżek, więc zawsze najlepiej zacząć od najtańszych. Dokładniej, będziemy chcieli szukać ścieżki powiększającej *o najmniejszym koszcie za jednostkę przepływu*, czyli minimalizującej sumę $\sum_e k(e)$. Nietrudno zauważyć, że bardzo to przypomina sformułowanie problemu najkrótszej ścieżki – właśnie tego będziemy używać. Wystarczy znaleźć w sieci rezydualnej najkrótszą ścieżkę z s do t, gdzie wagami są koszty $k(e)$. Nie możemy jednak zapominać o tworzonych przez nas krawędziach stowarzyszonych e': ponieważ przepchnięcie przez nie przepływu symbolizuje wycofanie przepływu z e (co zmniejszy sumaryczny koszt o $k(e)$ na wycofaną jednostkę), to powinniśmy przypisać $k(e') = -k(e)$.

Rozwiązanie Podsumowując, w pętli szukającej ścieżek powiększających powinniśmy priorytetyzować najkrótsze ścieżki, gdzie wagami są koszty. W sformułowaniu z budżetem K, wystarczy powtarzać szukanie ścieżek powiększających, brać jak najtańsze, wziąć jak najwięcej przepływu, by zmieścić się budżecie (respektując ograniczenia na Δf), po czym odpowiednio zmodyfikować dostępny budżet. Natomiast rozważając maksymalny przepływ o minimalnym koszcie (ten problem zwykle określa się Max-Flow Min-Cost) nie musimy zbytnio nic robić: wystarczy zwiększać przepływ, dopóki się da, wybierając najtańsze ścieżki powiększające. Ogólniej, da się szukać najtańszego przepływu danej wielkości analogicznie (w odpowiednim momencie przestajemy zwiększać przepływ).

Ujemne wagi Warto zwrócić uwagę, że w grafie, w którym szukamy najkrótszych ścieżek, są ujemne wagi! Można np. użyć SPFA (Shortest Paths Faster Algorithm, w uproszczeniu Bellman-Ford z kolejką) albo korzystać ze specyficznej postaci grafu. Warto zauważyć, że w początkowej sieci mogą się pojawić ujemne cykle. Problem wtedy będzie trudniejszy, ale da się go rozwiązać (trzeba uważać). Jednak jeżeli nie było ich wcześniej, to nie mogą się pojawić. Poza tym, warto wiedzieć, że jest sposób na takie zmodyfikowanie wag w grafie, aby nie było ujemnych wag – wystarczy jedno wywołanie Bellmana-Forda, potem można je aktualizować liniowo po każdej zmianie. Jest to opisane tutaj (potencjały):

`http://smurf.mimuw.edu.pl/node/1119`

Złożoność W sieciach o całkowitych przepustowościach, w zależności od zastosowanego sposobu szukania najkrótszych ścieżek, złożoność powyższego algorytmu to $O(fVE)$ (Bellman-Ford) bądź $O(VE + fE \log V)$ (Dijkstra z potencjałami).

4.3 „Obwód" optymalizacyjny

Jak na razie, przepływy przydawały się w rzeczach gdzie coś... przepływa. Albo się kojarzy. W każdym razie, zapomnieliśmy o Max-Flow Min-Cut – czas coś zrobić z minimalnymi przekrojami. Najpierw zinterpretujemy je na bardziej intuicyjny sposób.

Jak wiemy, *a-b* przekrój to podział wierzchołków na dwie grupy, że pierwsza zawiera a, a druga zawiera b. Oznaczmy grupy jako odpowiednio A i B. Jak wiadomo, wartość przekroju to suma po wagach krawędzi idących z A do B. Zróbmy tabelkę, która powie nam, kiedy płacimy za daną krawędź, w zależności od tego, w jakich grupach są jej końce (0 jeżeli płacimy):

$u \to v$	$v \in A$	$v \in B$
$u \in A$	1	0
$u \in B$	1	1

Co innego nam to przypomina, hm... A gdyby tak trochę logiki?

$p \Rightarrow q$	$q \equiv 1$	$q \equiv 0$
$p \equiv 1$	1	0
$p \equiv 0$	1	1

Możemy zinterpretować wierzchołki jako zdania logiczne, a krawędzie jako implikacje! Oznaczmy źródło jako $\mathbb{T}$, a ujście jako $\mathbb{F}$ (w celach niecnych). Niech grupami będą odpowiednio **1** oraz **0**. Teraz już widać, że wartość przekroju zwiększa się, gdy łamiemy implikację: $p \Rightarrow q$ jest fałszywe wtedy i tylko wtedy, gdy $p \equiv 1$ oraz $q \equiv 0$. Tak samo tutaj waga $u \to v$ jest doliczona do wartości, gdy u jest w grupie **1** z $\mathbb{T}$ (true), a v jest w **0** z $\mathbb{F}$ (false). Na podstawie tego możemy zamiast tego rozważyć zbiór implikacji, z których każda ma wagę, i chcemy tak przyporządkować zdaniom wartości logiczne, aby suma wag złamanych (fałszywych) implikacji była jak najmniejsza. Niech $p \overset{w}{\Rightarrow} q$ oznacza implikację od p do q wagi w.

Za pomocą takiego języka zapisywania przekrojów możemy zapisać kilka przydatnych schematów:

Nazwa	Zapis
(Zawsze prawdziwa) implikacja	$p \overset{\infty}{\Rightarrow} q$
Koszt fałszu	$\mathbb{T} \overset{w}{\Rightarrow} p$
Koszt prawdy	$p \overset{w}{\Rightarrow} \mathbb{F}$
Alternatywa (dowolne p_i daje q)	$\forall_i p_i \overset{\infty}{\Rightarrow} q$
Koszt nierównoważności	$p \overset{w}{\Rightarrow} q$, $q \overset{w}{\Rightarrow} p$

Niestety, za pomocą tego podejścia nie można sformułować negacji ani koniunkcji (AND). Koniunkcję zwykle można wyrazić „odwracając" stwierdzenie i mówiąc, że jeżeli $p \wedge q \Rightarrow r$, to zamiast tego $r \Rightarrow p$, $r \Rightarrow q$ (trzeba wtedy odwrócić też resztę rozumowania). Poza tym, należy zwrócić uwagę, że minimalny przekrój (czyli maksymalny przepływ) da nam przyporządkowanie, które minimalizuje sumę wag. Jeżeli chcemy maksymalizować sumę wag, to musimy przeformułować problem na minimalizację (ale powinno być to możliwe). Do tego wagi muszą być nieujemne.

4.4 Skojarzenia w grafie dwudzielnym

Często za pomocą maksymalnego przepływu można też rozwiązać problemy związane ze skojarzeniami. Za najprostszą wariację można podać sytuację, gdy mamy powiedziane, że jest x wierzchołków, które są połączone z $v_1, v_2, ..., v_k$. Wtedy można to załatwić konstruując prostą sieć przepływową i używając przepustowości x na odpowiedniej krawędzi. Jest to dobre ćwiczenie dla czytelnika. Do tematu skojarzeń powrócimy na innej kartce.

4.5 Prądy i inne takie

Na sam koniec rozważmy bardzo ciekawy koncept. Niektórym warunek zachowania przepływu:

$$\sum_{u \xrightarrow{e} v} f(e) = \sum_{v \xrightarrow{e} u} f(e)$$

Mógł przypominać pewien inny podobny koncept, dokładniej I prawo Kirchhoffa (suma natężeń prądu wpływających jest równa sumie wypływających):

$$\sum_i I_{in(i)} = \sum_i I_{out(i)}$$

Okazuje się, że istnieją metody konstruowania obwodów elektrycznych, które pomagają liczyć maksymalny przepływ w odpowiadającej sieci. Robi się to odpowiednio dopasowując opory na przewodach (przez które przechodzi natężenie): prąd będzie przepływał w sposób, który minimalizuje energię stanu (tudzież sumę mocy):

$$\sum_i P_i = \sum_i R_i I_i^2$$

W tych metodach manipuluje się wartościami oporu R_i, dopóki otrzymane I_i nie spełniają odpowiadających warunków na przepustowości (większy opór).

5 Zadanka

Główny problem z zadaniami na przepływy jest taki, że często trudno jest się domyślić, że chodzi właśnie o przepływy. Zwykle mogą nam to sugerować tylko niskie bądź średnie limity (a często może się okazać, że chodzi o algorytm zachłanny). Dlatego warto przerobić dużo przykładowych zadań!

5.1 [IndiaHacks 2016] Delivery Bears

Treść Chcemy rozwiązać problem maksymalnego przepływu w danej sieci, ale z drobną modyfikacją: każda ścieżka powiększająca ma przenosić dokładnie x przepływu (gdzie $x \in \mathbb{R}$ to pewna wartość, którą chcemy zmaksymalizować), a ścieżek powiększających ma być k (które jest podaną stałą).

Ograniczenia $n \leqslant 50$, $m \leqslant 500$, $k \leqslant 10^5$, błąd $\epsilon = 10^{-6}$

Rozwiązanie Chcemy sprowadzić to do klasycznego problemu maksymalnego przepływu. Zacznijmy od wyszukiwania binarnego po wartości x (widać, że jeżeli da się x, to da się też każdy $x_1 < x$). Aby pozbyć się wymagania o przepływie na jednej ścieżce powiększającej, dla każdej krawędzi zastąpmy jej przepustowość c przez $\lfloor \frac{c}{x} \rfloor$ (jest to największa krotność x, jaką możemy nią przepuścić). Następnie znajdźmy maksymalny przepływ. Po tym wystarczy przemnożyć wszystkie przepływy przez x, aby otrzymać przepływ w oryginalnej sieci. Jeżeli jest on równy co najmniej kx, to x może być wynikiem. Powtarzamy iteracje wyszukiwania binarnego, dopóki nie osiągniemy satysfakcjonującej precyzji.

5.2 [Codeforces Round #212] Petya and Pipes

Treść Ponownie mamy do czynienia z siecią przepływową, i mamy do dyspozycji k ruchów. W każdym ruchu możemy zwiększyć przepustowość istniejącej krawędzi o 1. Naszym zadaniem jest zmaksymalizować maksymalny przepływ w sieci po wykonaniu naszych ruchów, i wypisać jego wartość.

Ograniczenia $n \leqslant 50$, $m \leqslant n(n-1)$, $k \leqslant 10^3$

Rozwiązanie Dosyć łatwo zauważyć, że mamy tutaj do czynienia z dwoma surowcami: ruchami i przepływem, co sugeruje, żeby spróbować użyć Max-Flow Min-Cost. Pomysł jest prosty: zwiększenie przepustowości na krawędzi e może być symulowane istnieniem krawędzi o koszcie 1 (koszt symbolizuje wykonanie ruchu) o takich samych końcach jak e. Ponieważ przepustowość na danej krawędzi możemy zwiększać dowolnie wiele razy (zapominając na chwilę o globalnym limicie k), możemy nadać nowej krawędzi przepustowość ∞. Istniejące wcześniej krawędzie mają koszt 0. W zmodyfikowanej sieci szukamy maksymalnego przepływu o koszcie $\leqslant k$ – jest to nasz wynik.

5.3 [XVII OI, etap III] Mosty

Treść Wybieramy się wycieczkę rowerową w San Bajcisku, złożonym z archipelagu n wysp połączonych m (dwukierunkowymi) mostami. W zależności od tego, w którą stronę jedziemy mostem, odczuwamy różną siłę wiatru: l_i lub r_i. Chcemy znaleźć taki cykl Eulera (w pojęciu nieskierowanym – każdym *mostem* przejeżdżamy dokładnie raz), że maksymalna odczuwana siła wiatru jest jak najmniejsza – lub stwierdzić, że cykl nie istnieje.

Ograniczenia $n \leqslant 10^3$, $m \leqslant 2 \cdot 10^3$

Rozwiązanie Najpierw sprawdźmy, czy cykl Eulera istnieje: każda wyspa (wierzchołek) musi mieć stopień parzysty. Minimalizowanie maksimum od razu sugeruje, by zacząć od wyszukiwania binarnego po wyniku: maksymalnej sile wiatru ω. Gdy to zrobimy, mosty (krawędzie) można podzielić na trzy grupy:

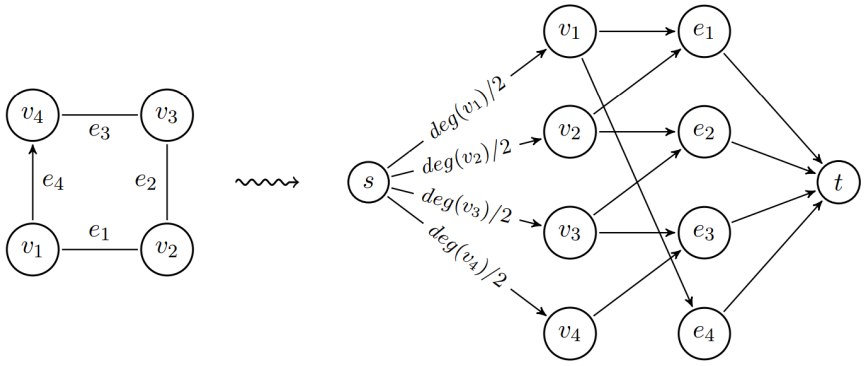
1. Takie, którymi nie da się przejechać: $l_i, r_i > \omega$. W takim wypadku na pewno cykl Eulera nie istnieje, bo nie przejedziemy tym mostem. Nie rozpatrujemy dalej tej grupy.
2. Krawędzie, dla których musi przejechać w jedną bądź drugą stronę (dokładnie jedno z l_i, r_i jest $\leqslant \omega$).
3. Krawędzie, którymi możemy przejechać w obie strony.

Zatem widzimy, że musimy jakoś zdecydować o kierunku przejścia krawędziami z grupy 3 – chcemy tak *zorientować* krawędzie naszego grafu nieskierowanego, by powstały graf skierowany był eulerowski. Przypomnijmy sobie, skąd się bierze warunek na istnienie cyklu Eulera: do każdego wierzchołka musimy wejść tyle samo razy, ile wyjdziemy: oba z tych zdarzeń zajdą $\frac{\deg u}{2}$ razy.

Przeformułujmy trochę proces orientacji grafu. Gdy skierowujemy krawędź $u \to v$, to zwiększamy stopień wyjściowy u (o_u) oraz wejściowy v (i_v) o 1. Na koniec chcemy, by dla każdego wierzchołka v zachodziło $i_v = o_v$, bo wtedy wiemy, że orientacja spełnia warunek na cykl Eulera. Ale $i_v + o_v = \deg v$, zatem wystarczy, by $o_v = \frac{\deg v}{2}$. Jedyne co nam teraz pozostaje, to skonstruować odpowiednią sieć przepływową.

W naszej sieci będą wierzchołki odpowiadające wyspom – $v_1, ..., v_n$ – oraz krawędziom – $e_1, ..., e_n$. Chcemy, aby każdy wierzchołek wybrał sobie podzbiór $\frac{\deg v_i}{2}$ krawędzi, którymi go opuścimy – w ten sposób zapewnimy warunek $i_{v_i} = o_{v_i}$. Sieć przepływową konstruujemy następująco: tworzymy krawędzie $e_i \xrightarrow{1} t$ (krawędź może być wybrana najwyżej raz), $s \xrightarrow{\deg v_i/2} v_i$ (warunek cyklu). Pozostały krawędzie umożliwiające wybór: jeżeli e_i może być skierowane „od" v_i, to tworzymy krawędź $v_i \xrightarrow{1} e_i$. Jeżeli przyporządkowanie zostało znalezione, maksymalny przepływ to m.[2]

[2] Poniższy obrazek pochodzi z niebieskiej książeczki.



W otrzymanej sieci zachodzi $|V| = O(n+m)$ i $|E| = O(n+m)$, ale $f \leqslant m$, zatem złożoność jest rzędu $O(m(n+m))$. To zadanie ma dosyć długie rozumowanie, ale w gruncie rzeczy sprowadzamy problem do skojarzeń, które często pojawiają się w problemach związanych z rozkładem grafu na cykle.

5.4 [Codeforces Round #147] Build String

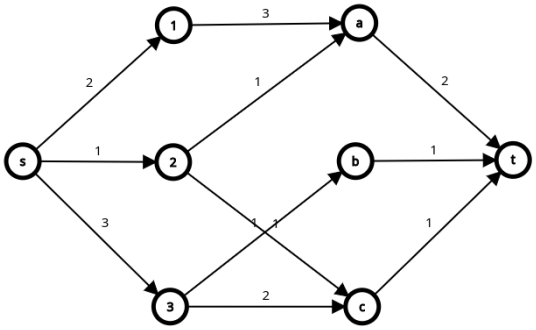
Treść Mamy słowniczek wyrazów $s_1, ..., s_n$ i chcemy ukraść z nich literki (z dowolnych miejsc, w dowolnej kolejności), by otrzymać słowo t. Żeby nie było za łatwo, koszt zabrania literki z i-tego wyrazu to i, a dodatkowo nie można zabrać więcej niż a_i literek z tego wyrazu.

Ograniczenia $n \leqslant 100$, $a_i \leqslant 100$, $|t|, |s_i| \leqslant 100$, $\sigma = 26$ (małe litery łacińskie)

Rozwiązanie Na razie zapomnijmy o kosztach – zróbmy sieć przepływową, która pozwoli nam znaleźć konstrukcję t przy ograniczeniach a_i.

- Stwórzmy wierzchołek u_i „spinający" akcje zabrania pewnej literki z s_i: możemy to zrobić co najwyżej a_i razy, zatem $s \xrightarrow{a_i} u_i$.
- Potem ograniczają nas tylko liczebności poszczególnych literek, więc zróbmy wierzchołki do zabrania literki α: v_α. Dla każdego i tworzymy krawędzie $u_i \to v_\alpha$ o takiej przepustowości, ile jest liter α w s_i.
- Na sam koniec wystarczy zapewnić, że weźmiemy odpowiednią liczbę liter: do źródła przepuszczamy tylko tyle liter α, ile jest ich w t: $v_\alpha \to t$.

Na obrazku cyfry to wierzchołki u_i, a litery to v_α.



Aby załatwić koszty wystarczy tylko nałożyć koszt $i\$$ na krawędź $s \xrightarrow{a_i} u_i$. Mamy $|V| = O(n + \sigma)$, $|E| = O(n\sigma) = O(n^2)$, $f \leqslant |t|$.

5.5 [AMPPZ 2011] Iloraz inteligencji

Treść Mamy n studentów matematyki i m informatyki (każdy studiuje dokładnie jeden kierunek). Znamy ilorazy inteligencji studentów, oznaczane literami a_i (dla i-tego studenta matematyki) bądź b_j (j-ty informatyki). Znamy też znajomości między nimi: znają się wszyscy studenci tego samego kierunku oraz istnieje też k par znajomych z różnych kierunków. Naszym zadaniem jest znaleźć podzbiór wszystkich studentów o maksymalnej sumie ilorazów inteligencji, taki, że każda para studentów się zna.

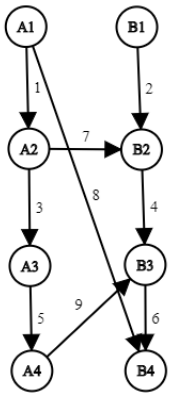
Ograniczenia $n, m \leqslant 400$, $k \leqslant n \cdot m$

Rozwiązanie Nietrudno zauważyć, że w zadaniu mamy do czynienia z grafem dwudzielnym. W postawionym problemie można zauważyć wariację problemu największej kliki, lecz jest ona tutaj trochę zmodyfikowana: chcemy znaleźć biklikę, czyli taki podzbiór wierzchołków, że każda para wierzchołków z różnych stron grafu jest połączona krawędzią.

Kliki łatwo przedstawić logicznie: jeżeli weźmiemy wierzchołek u, a nie jest on połączony z v, to nie możemy wziąć też v. Jeżeli $[u]$ oznacza wzięcie u, to $[u] \Rightarrow \neg[v]$. To sugeruje, by wykorzystać Max-Flow Min-Cut (w interpretacji obwodu). Trzeba uważać, żeby nie wpaść pułapkę stosowania *tylko* zmiennych rodzaju „czy wzięliśmy studenta u", bo w obwodach nie możemy skonstruować negacji. Skorzystajmy z tego, że graf jest dwudzielny: niech $[i]$ oznacza, że wzięliśmy i-tego studenta matematyki, a $[[j]]$, że **nie** wzięliśmy j-tego informatyki. W takim układzie mamy $\mathbb{T} \overset{a_i}{\Rightarrow} [i]$ (jeżeli nie wzięliśmy i, to tracimy a_i IQ) oraz $[[j]] \overset{b_j}{\Rightarrow} \mathbb{F}$ (jeżeli nie wzięliśmy j, to tracimy b_j). Pozostaje odzwierciedlić znajomości: $[i] \overset{\infty}{\Rightarrow} [[j]]$, jeżeli nie ma znajomości między i oraz j. Łatwo zauważyć, że odjęcie minimalnego $\mathbb{T}$-$\mathbb{F}$ przekroju w odpowiedniej sieci przepływowej od sumy IQ wszystkich studentów da nam pożądany wynik.

5.6 [Educational Codeforces Round #34] Yet Another Maxflow Problem

Treść Mamy sieć przepływową złożoną z dwóch „kolumn": $(A_1, ..., A_n)$ oraz $(B_1, ..., B_m)$. Dla każdego i istnieje krawędź $A_i \xrightarrow{x_i} A_{i+1}$ oraz $B_i \xrightarrow{y_i} B_{i+1}$. Istnieje też k krawędzi postaci $A_{u_i} \xrightarrow{z_i} B_{v_i}$. Należy umieć modyfikować x_i (przepustowość na krawędziach $A_i \to A_{i+1}$) oraz odpowiadać na zapytania o maksymalny przepływ w sieci ze źródła A_1 do ujścia B_m.



Ograniczenia $n, m, q \leqslant 2 \cdot 10^5$, przepustowości do 10^9

Rozwiązanie Dwie rzeczy są wyjątkowe w tym zadaniu: po pierwsze limity są zdecydowanie zbyt duże jak na przepływy, a po drugie sieć przepływowa jest bardzo specyficzna. Co najmniej jedna z tych obserwacji powinna nam zasugerować, by pomyśleć o innym sposobie liczenia maksymalnego przepływu: Max-Flow Min-Cut.

Rozważmy[3] zatem A_1-B_m przekrój, dzielący wierzchołki na zbiór S ($A_1 \in S$, $B_m \notin S$) oraz pozostałe, i rozważmy wierzchołki zawierające się w S. Kluczowa obserwacja jest następująca: weźmy minimalny przekrój S, a następnie wybierzmy $A_i \in S$ takie, że $A_{i+1} \notin S$. Przypuśćmy, że istnieje $j > i$ takie, że $A_j \in S$ (i wybierzmy najmniejsze możliwe). Wtedy możemy skonstruować przekrój $S_1 = S \setminus \{A_j\}$, który będzie miał niewiększą wartość. Uzasadnienie tego faktu jest dosyć proste: ponieważ j jest najmniejsze, to $A_{j-1} \notin S$, więc na pewno w S_1 krawędź $A_{j-1} \to A_j$ nie jest przecięta (czyli prowadzi z wierzchołka w S poza S). Do tego krawędź $A_j \to A_{j+1}$, ani żadna z $A_j \to B_k$ nie jest przecięta, bo $A_j \notin S_1$. Po tej zmianie, jedyne co się mogło stać, to że niektóre krawędzie przestały być przecinane, zatem wartość przekroju S_1 nie mogła się zwiększyć. Rozumując podobnie dla B, przypuśćmy, że istnieje indeks i, taki że $B_i \in S$, ale $B_{i-1} \notin S$. Wtedy skonstruowanie $S_1 = S \cup \{B_{i-1}\}$ nie może zwiększyć wartości przekroju, co możemy uzasadnić tak jak wcześniej.

A jaki z tego wniosek? Widać, że z tego rozumowania wynika, że minimalny przekrój możemy znaleźć rozpatrując tylko S zawierające **prefiksy** kolumn: $(A_1, A_2, ..., A_a)$ oraz $(B_1, B_2, ..., B_b)$. Jest to kluczowa obserwacja i pewnie najtrudniejsza część zadania.

Aby dokończyć rozwiązanie, opiszmy wartości przekroju względem wielkości prefiksów A oraz B (odpowiednio a, b). Wartość przekroju jest sumą:

$$x_a + \underbrace{y_b + \texttt{between}(a, b)}_{\text{const}}$$

Gdzie $\texttt{between}(a, b)$ to suma po przeciętych krawędziach między A i B, czyli suma po przepustowościach z wszystkich krawędzi $A_i \xrightarrow{z} B_j$ dla $i \leqslant a$ oraz $j > b$.

[3]Uwaga, dowód konstrukcyjny. Niecierpliwy czytelnik może pominąć ten akapit i przejść do wniosku.

Pozostaje zauważyć, że jedyna wartość, która może się tutaj zmienić (po modyfikacjach) dla ustalonego (a, b) to x_a. Co za tym idzie, dla każdego a możemy wyznaczyć optymalną wartość b, dla której reszta sumy jest zminimalizowana. Wtedy wartości dla ustalonych a możemy przechowywać w drzewie przedziałowym, a przy zmianie x_a wystarczy też trochę zmienić wartość w odpowiednim liściu.

$$\min_{a,b} x_a + y_b + \texttt{between}(a, b) \overset{\text{fix } a}{=} x_a + \min_b y_b + \texttt{between}(a, b)$$

Szukanie optymalnych b dla ustalonych a możemy przeprowadzić rozważając kolejno $a = 1, 2, ..., n$ i utrzymując na drzewie przedziałowym koszty dla poszczególnych b (wzięcie pod uwagę wszystkich krawędzi $A_a \to B_j$ to kwestia dodania na sufiksie). Początkowo w liściach drugiego drzewa są wartości y_b. Szczegóły wykraczają poza zakres kółka o przepływach `:)`. Całość można rozwiązać w $O((n + m) \log(n + m) + q \log n)$.

5.7 [POJ (Peking University)] Intervals

Treść Mamy n przedziałów, każdy z nich jest obustronnie otwarty i i-ty obejmuje (a_i, b_i) oraz ma wagę w_i. Należy znaleźć podzbiór przedziałów o maksymalnej sumie wag, taki, że nie ma punktu, który jest pokryty przez więcej niż k przedziałów.

Ograniczenia $n, k \leqslant 200$, $a_i, b_i, w_i \leqslant 10^5$

Rozwiązanie Zacznijmy od przeformułowania problemu: globalny warunek „co najwyżej k przedziałów" jest trudny do sformułowania w języku przepływów (jak można się łatwo przekonać). W tym celu zauważmy następujący fakt, który może przynosić na myśl twierdzenie Dilwortha:

Wszystkie punkty na osi są pokryte przez nie więcej niż k przedziałów wtedy i tylko wtedy, gdy przedziały można podzielić na co najwyżej k rozłącznych grup, z czego każda zawiera pewien podzbiór rozłącznych (nie przecinających się) przedziałów.

Równoważność jest prosta do pokazania: dla ($\Rightarrow$) wystarczy rozważyć prosty algorytm zachłanny przydzielający przedziały do grup idąc od lewej do prawej po osi. W ($\Leftarrow$) poprawność jest oczywista (w danym punkcie może być co najwyżej jeden przedział z danej grupy, których jest k).

Chcemy za pomocą przepływów konstruować opisany podział na grupy – na razie zapomnijmy o wagach. Wyślemy k jednostek przepływu, a każda, idąc swoją ścieżką powiększającą, określi nam wybrane do grupy przedziały. Posortujmy przedziały leksykograficznie (czyli rosnąco po początkach, a jeżeli początki są takie same, to po końcach). Pomysł jest następujący: jednostka przepływu (reprezentująca przydział do grupy) będzie po kolei szła po przedziałach, i jeżeli zdecyduje się jakiś wziąć, to przeskoczy wszystkie, które ten przedział przecina. Korzystamy tutaj z tego, że gdy posortujemy przedziały leksykograficznie,

to dany przedział przecina następujące przedziały tylko na **spójnym** podciągu ciągu przedziałów – więc możemy łatwo ominąć ten spójny podciąg.

Wierzchołki w sieci reprezentują przedziały, a krawędzie wzięcie bądź ominięcie przedziału. Oznaczmy posortowany ciąg przedziałów jako $I = (I_1, I_2, ..., I_n)$. Czas określić, jak dokładnie ma wyglądać sieć:

- Źródłem jest I_1, a ujściem I_n.
- Istnieje krawędź $I_i \xrightarrow{\infty} I_{i+1}$, reprezentująca nie wzięcie i-tego przedziału do grupy.
- Po wzięciu I_i chcemy przeskoczyć do najwcześniejszego leksykograficznie przedziału występującego po nim, który nie przecina I_i. Oznaczmy jego indeks jako $\mathcal{N}(i)$. Istnieje krawędź $I_i \xrightarrow{1} I_{\mathcal{N}(i)}$ (przepustowość to 1, bo dany przedział może być wzięty tylko raz).

Żeby załatwić wagi należy (jak zwykle) wykorzystać Max-Flow Min-Cost. Krawędzie symbolizujące wzięcie I_i otrzymają koszt $-w_i$. Otrzymamy sieć wielkości liniowo zależnej od n. Wystarczy w niej znaleźć najtańszy przepływ o wartości co najwyżej k, i zwrócić jego zanegowany koszt.

5.7.1 Twierdzenie Dilwortha

Odbiegając trochę od tematu, twierdzenie Dilwortha uogólnia zastosowany fakt dla dowolnych obiektów matematycznych, na których umiemy zdefiniować częściowy porządek $\preceq$ (to oznacza, że mogą istnieć pary elementów, których nie da się porównywać) – relację zwrotną, przechodnią i antysymetryczną. Weźmy częściowo uporządkowany zbiór. *Łańcuch* to taki jego podzbiór, że wszystkie elementy można porównać ze sobą (czyli można zrobić łańcuszek $x_1 \preceq x_2 \preceq ... \preceq x_d$). Natomiast *antyłańcuch* to podzbiór taki, że nie można porównać żadnej pary elementów, które się w nim znajdują. Oto treść twierdzenia:

Twierdzenie (Dilwortha)**.** *Rozmiar największego antyłańcucha w zbiorze to minimalna liczba rozłącznych łańcuchów, na które można podzielić cały zbiór.*

Fakt z zadania wynika, gdy jako $A \preceq B$ zdefiniujemy „czy przedziały A i B się nie przecinają oraz A jest na lewo od B". Inny przykład zastosowania tego twierdzenia jest związany z ciągami oraz podciągami rosnącymi i malejącymi. Dla ciągu $(a_1, a_2, ..., a_n)$ zdefiniujmy porównanie $(i) \preceq (j) \Leftrightarrow i \leqslant j \wedge a_i \leqslant a_j$. Wtedy łańcuchy to podciągi niemalejące, a antyłańcuchy to podciągi malejące. Otrzymujemy, że długość najdłuższego podciągu malejącego to minimalna liczba podciągów rosnących, którymi można pokryć ciąg.

Jako że jest to bardziej temat matematyczny (aczkolwiek kombinatoryka przydaje się w informatyce), odsyłam do innych źródeł:

- `https://en.wikipedia.org/wiki/Dilworth%27s_theorem`
- `https://brilliant.org/wiki/dilworths-theorem/`

5.8 [Forethought Future Cup 2019] Zoning Restrictions

Treść Na ulicy jest n działek w rzędzie, na których chcemy postawić budynki o wysokościach od 0 (czyli brak budynku) do h (najwyższy możliwy budynek). Chcemy, żeby budynki były jak najwyższe – budynek wysokości a daje nam zysk a^2. Jednakże, miasto nałożyło m ograniczeń budowlanych: i-te z nich mówi, że jeżeli wśród działek od l_i-tej do r_i istnieje budynek wyższy od x_i, to należy zapłacić (jednorazową) karę c_i. Znajdź maksymalny możliwy zysk!

Ograniczenia $n, h, m \leqslant 50$, $c_i \leqslant 5 \cdot 10^3$

Rozwiązanie Warunek „jeżeli wśród działek..." podpowiada podejście do problemu używając Max-Flow Min-Cut. Stwórzmy sieć, w której wierzchołki będą reprezentowały zdania „czy budynek na i-tej działce jest wysokości co najmniej a" – $[w_i \geqslant a]$ ($a \in [0, h]$) – oraz „czy zapłaciliśmy j-tą karę" – $\{j\}$.

- Każdy budynek musi być wysokości co najmniej 0, więc $\mathbb{T} \overset{\infty}{\Rightarrow} [w_i \geqslant 0]$. Poza tym musimy jakoś nauczyć obwód, jak działa porównywanie, mówiąc, że $[w_i \geqslant x + 1] \overset{\infty}{\Rightarrow} [w_i \geqslant x]$.
- Tutaj jest nietrywialna część: Skorzystamy z tego, że zyski są rosnące, i minimalny przekrój będzie minimalizował sumę nieotrzymanego zysku. Dokładniej, jeżeli $[w_i \geqslant a]$ jest prawdziwe, to z tej racji dodatkowo otrzymalibyśmy $a^2 - (a-1)^2 = 2a - 1$ (w porównaniu do $[w_i \geqslant a - 1]$). Zatem $\mathbb{T} \overset{2a-1}{\Rightarrow} [w_i \geqslant a]$.
- Teraz trzeba zająć się łamaniem ograniczeń: i-ty dla każdego $l_j \leqslant i \leqslant r_j$ ma wymuszoną implikację $[w_i \geqslant x_j + 1] \overset{\infty}{\Rightarrow} \{j\}$.
- Na koniec zostaje płacenie kar: $\{j\} \overset{c_j}{\Rightarrow} \mathbb{F}$.

Podsumowując: $|V| = O(nh + m)$, $|E| = O(nh + m + nm) = O(n(h + m))$, $f = O(nh^2)$.

6 Uwagi implementacyjne

Jest kilka podejść do implementacji. Przede wszystkim, zwykle utrzymuje się tylko sieć rezydualną – to znaczy utrzymujemy początkową sieć przepływową i dla każdej krawędzi znamy jej przepustowość rezydualną $c_f(e)$, do tego dla każdej krawędzi tworzymy krawędź stowarzyszoną e', taką że początkowo $c_f(e') = 0$ (oryginalna krawędź jest towarzyszem stowarzyszonej). Gdy przepychamy przepływ przez jedną z krawędzi, zmniejszamy jej przepustowość rezydualną, ale o tyle samo zwiększamy przepustowość krawędzi stowarzyszonej. W ten sposób przepychanie przez obie krawędzie (czyli w obie strony) działa na dokładnie tej samej zasadzie.

Należy jednak zdecydować, jak dokładnie będziemy szybko sprawdzać obecną przepustowość rezydualną krawędzi oraz jej krawędzi stowarzyszonej – to możemy rozwiązać inaczej, w zależności od tego, czy pojawiają się multikrawędzie. Jednak uniwersalnym sposobem jest przechowywanie list sąsiedztwa. W liście sąsiedztwa mogą „mieszkać" krawędzie (jako struktury), i wystarczy znać jej końce, obecną przepustowość i indeks krawędzi stowarzyszonej w odpowiedniej liście. Istnieją także podejścia oparte na wskaźnikach. Poza tym krawędzie mogą też mieszkać w globalnej tablicy (empirycznie, może być to wolniejsze), a w listach sąsiedztwa możemy przechowywać jedynie ich indeksy. Jeszcze innym sposobem jest przechowywanie globalnej tablicy `f[u][v]`, symbolizującej obecny przepływ między dwoma wierzchołkami – w tym wypadku niezbyt działają multikrawędzie (szczególnie przy implementacji Max-Flow Min-Cost).

Zasada co pisać:

1. Jeżeli limity są tak małe, że złożonościowo wejdzie Edmonds-Karp, albo graf jest wyjątkowo specyficzny, to można go napisać.
2. W przeciwnym wypadku: przepychać Dinica, gdy testy wydają się mocne to użyć Dinica ze skalowaniem.
3. Jeżeli chodzi o Max-Flow Min-Cost: analogicznie, jeżeli znacząco wychodzimy poza limity to można pisać Dijkstrę z potencjałami (ale to niekoniecznie musi być tak dobry pomysł jak Dinic zamiast Edmondsa-Karpa)

Moje implementacje w zadaniach (z komentarzami):

- Dinic w zadaniu FASTFLOW (ze SPOJa): [klik]
- Max-Flow Min-Cost w zadaniu Machine Programming: [klik]

7 *Zadanka do kminienia*

Do niektórych zadań jest przetłumaczona treść z:

- $\mathcal{M}$ – Kartki Marka Sommera: `https://mareksom.w.staszic.waw.pl/kolko/2013-2014/kartki/10.06.2014.pdf`
- $\mathcal{T}$ – Listy zadań Tymoteusza Wiśniewskiego: `http://students.mimuw.edu.pl/~tw418404/przeplywy.pdf`

Zadania:

- [Google Code Jam 2014 – Round 2] $\mathcal{M}$ **Don't Break The Nile**
- [Codeforces Round #290] $\mathcal{T}$ Fox and Dinner
- [VK Cup 2012] Machine Programming
- [Finał Google Code Jam 2009] $\mathcal{M}$ **Wi-fi Towers**
- [XVIII OI, etap III] Konkurs programistyczny
- **Bejsbol** (klasyczny problem) – Jest n drużyn bejsbolowych, i-ta z nich wygrała już w_i meczów i zagra z j-tą jeszcze $r_{i,j}$ meczów. Drużyna jest *wyeliminowana*, jeżeli nie ma takiego scenariuszu, w którym kończy ona rozgrywki na pierwszym miejscu (z największą liczbą zwycięstw). Przyjmijmy, że może być *ex aqueo*. Stwierdź, czy k-ta drużyna jest wyeliminowana, lub podaj scenariusz w którym jest na pierwszym miejscu.
- [VN SPOJ] Fast Flow – zadanie z wietnamskiego SPOJa, które ma pesymistyczne testy na algorytmy przepływowe. Trzeba użyć lepszego algorytmu (np. Dinica ze skalowaniem). Więcej o przepychaniu przepływów można przeczytać na [tym blogu na CodeForces]. Zaznaczam, że wygląda na to że autor ma inne czasy niż podaje na tym zadaniu (jego kod daje około 0.09s zamiast 0.00s).