## Мрежи. Максимален поток.

## Въведение

Често срещан проблем в практиката е да се използва максималната пропускливост на мрежа(пътна, компютърна, система от тръби). Примерен проблем е употребата на пътните артерии от фабрика, която иска да закарва максимално количество продукция за единица време до складовете си.

## Дефиниция на проблема

Искаме да прекараме възвможно най-голямо количество товари от един град до друг като използваме съществуващата пътна мрежа. Дадена ни е мрежата – насочен граф G(V, E), с теглова функция по ребрата “**c**” : V \* V=R+0(реално неотрицателно число) – капацитет на реброто. Искаме да определим какво най-голямо количество стоки може да се закара от “**s**” (стартовия град) до “**t**”(крайния град), като за функцията “c” можем да считаме, че е броят камиони, които могат да преминат от един град до друг за един ден. За удобство считаме, че всяко ребро от графа лежи на път от “**s**” до “**t**”.

Това е често срещан проблем с практическо приложение – проблема за максималния поток. За да решим задачата ще дефинираме следните понятия:

**Поточна мрежа**:

Ориентиран граф G(V, E). означаваме: v1,v2,..,vn - върховете на графа

**Капацитетна функция**:

c(u, v) >= 0, за всяко u, v от V, като c(u, v) = 0, ако не съществува ребро от u до v. (капацитета на дадено ребро.)

**поток**:

Ако f е реална функция дефинирана върху V \* V, f се нарича поток ако са изпълнени:

1) f(u,v) = -f(v,u)

2) За всяко v от V\{s,t}: сумата f(v, v1) + f(v, v2) + … + f(v, vn) = 0

3) f(u, v) <= c(u, v)

Тоест функцията f представлява насоченият поток, който тече по дадено ребро.

**обем на поток** е сумата на f(s, v1) + f(s, v2) + … + f(s, vn). Тази сума е същата като f(v1, t) + f(v2, t) + … + f(vn, t). Означаваме я с |f|.

Тогава отговорът на задачата ще бъде максималния възможен |f|.

Ще дефинираме и понятието **остатъчна мрежа**, което представлява поточната мрежа, но с останали само ребрата, по които все още може да се пропуска поток. Остатъчната мрежа се дефинира върху същия граф G(V, E), но се въвежда нова функция – **остатъчeн капацитет**: cf(u, v) = c(u, v) – f(u, v). Тогава Gf(V, E’), където E’ са онези ребра за които cf(u, v) > 0. Ще ги наричаме **ребра с остатъчен капацитет**.

Ще дефинираме и понятието **път, подлежащ на увеличение** – път от s до t, който може по който може да се увеличи текущия поток. Това е път с ребра само от Gf – остатъчната мрежа.

## Теория

(s,t) срез (*A*,*B*) е разделяне на множеството на върховете на две множества A, B  (A \cap B = \emptyset , A \cup B = V)така, че s \in A, t \in B

капацитет на срез е C(A,B) = \sum_{a \in A,b \in B} c(a,b)т.е. сумата от капацитетите на всички ребра, свързващи множествата.

**Теорема за минималния срез и максималния поток**

Ако f е поток, следните три са еквивалентни:

1) f e максимален поток

2) Съществува (s,t) срез (*A*,*B*) с *C*(*A*,*B*) = | *f* |

3) Няма път от от s до t в Gf.

**Алгоритъм за намиране на минимален срез (минимален разрез, minimum cut)**

Намираме максимален поток. Намираме достижимите от s върхове в Gf. Тези върхове образуват едно множество A, останалите множeство B. Всички ребра свързващи връх от A с връх от B образуват минималния срез.

## Алгоритъм

Ето го и методът за намиране на максимален поток:

1) Инициализираме f[u][v] = 0, за всяко u и v.

2) Докато има път, подлежащ на увеличение, пускаме максималния възможен поток по този път и актуализираме остатъчната мрежа. (Това е най-малкия остатъчен капацитет на ребро от пътя.)

3) Изчисляваме сумата f(u1, t) + f(u2, t) + … + f(un, t) – това е отговора на задачата.

## Псевдо код

Ще напишем и малко псевдо-код, за да можем да изчислим сложността на получения алгоритъм.

for ( за всяко u, v)

{

f[u][v] = 0;

cf[u][v] = 0;

}

while (има път path от s до t в Gf)

{

increment\_flow = min(cf[u][v], където u, v са от пътя path);

for (за всяко ребро (u, v) от пътя path)

{

f[u][v] += increment\_flow;

f[v][u] = -f[u][v];

cf[u][v] = c[u][v] – f[u][v];

}

}

Res = sum(f[u][t]), за u от 1 до n.

## Сложност

Сложността на алгоритъма зависи от това колко пъти се изпълнява цикъла while – нека това е W. При всяко завъртане на while цикъла се правят О(М + N) стъпки, където М e броят на ребрата в графа, a N на върховете. М стъпки за откриването на път подлежащ на увеличение чрез търсене в ширина и Н стъпки за актуализиране на пътя. Тогава сложността е О(W\*(М+N)), тоест O(W\*M).

Тежестта на алгоритъма се свежда до това да сведем броя на завъртанията на цикъла до минимум. За целта можем винаги да пускаме обхождане в ширина, което ни дава най-краткия път по брой ребра от s до t в остатъчната мрежа. Чрез такова избиране на път подлежащ на увеличаване можем да докажем, че цикълът while се изпълнява O(M\*N) пъти. Ще иползваме термина критично ребро – това е реброто, което дава стойността на increment\_flow при актуализирането на път. Всеки път има поне едно такова ребро (като в най-лошия случай има точно едно). Всяко ребро може да е критично най много N/2 – 1 пъти. Тогава M ребра, по N/2-1 пъти – O(M\*N) пъти най-много се изпълнява while цикъла. Ребро може да стане критично най-много N/2 – 1 пъти, защото след като стане критично изчезва от графа. Появява се чак след като от него се отнеме потоко – обратното му участва в някои път подлежащ на увеличаване. С този алгоритъм можем да постигнме сложност О(N\*M^2), но съществуват и алгоритми, които работят по-бързо.

## По-добър алгоритъм

ще покажа и един по-добър алгоритъм за намиране на max-flow. Може да го намерите в Интернет под името Dinitz. Алгоритъма се състои в следното:  
  
**1**) пускаме bfs от sink, която се движи по обратните ребра (т.е за да стигне от i до j трябва Gf стойността на реброто от j до i да е ненулева), като по този начин за всеки връх изчисляваме дължината на най-краткото му разстояние до sink-а. Ако при това обхождане не посетим source - алгоритъма приключва (тъй като няма път в Gf от source до sink)  
  
**2**) Пускаме dfs от source (движи се по нормалните ребра т.е за да стигне от i до j трябва Gf стойността на реброто от i до j да е ненулева) като допускаме движение само през ребра, с които намаляваме разстоянието до sink-а.  
Ако достигнем sink:  
**2.1**) правим source и sink неизползвани (used[sink] = used[source] = 0), намираме най 'тясното' ребро по пътя които сме намерили - нека то е с големина CurrentFlow. Увеличаваме потока с CurrentFlow по намерения път от source до sink (отразяваме промените и в графа Gf). Повтаряме от точка 2  
Ако не достигнем sink:  
**2.2**) правим непосетени всички върхове. Повтаряме от точка 1.  
  
  
Интересното при този алгоритъм е, че сложността на dfs-тата (точка 2) между 2 пускания на bfs (точка 1) е колкото сложността на 1 dfs. Това е така, защото всеки връх (освен source и sink) се обхожда максимум веднъж - ако 1 dfs направи даден връх използван (used[a] = 1), то следващото dfs няма да мине през него. Забележете че 'изчистването' на масива used, не се прави след всяко dfs.  
Сложността на този алгоритъм за matching е O(\sqrt{n}m)

## Разширяване на проблема

Разгледаният по горе проблем се занимава само с един източник(source - t) и едно място където отиват стоките(sink - t), но това лесно може да се промени. Добавя се SuperSource и SuperSink и се разширява графа така че от тях да излизат/влизат ребра към/от останалите източници/складове.

## Материали

Статията е подготвена осново по материали от "Introduction to Algorithms" 2ed. MIT Press и от <http://homepages.cwi.nl/~lex/files/agtco.flows.pdf>. За повече материали и за подробни и доста разбираеми доказателства на по горните твърдения погледнете там(особено в Intro-то). Взето от <http://judge.openfmi.net/wiki/index.php/%D0%9C%D0%B0%D0%BA%D1%81%D0%B8%D0%BC%D0%B0%D0%BB%D0%B5%D0%BD_%D0%BF%D0%BE%D1%82%D0%BE%D0%BA>".

Дадената задача в този кръг може да се разгледа като обобщение на задача 5 от 2001-02 година и задача 4 от 2003-04 година, които се свеждаха до намиране на максимален поток в граф. Предлагаме на тези, които не са запознати с този проблем, да прочетат първо анализите на горните две задачи на сайта на конкурса.

Крайния ориентиран граф (без примки) *G(V,E)*, по чиито ребрата са дефинирани функции *cap* и *cost* с положителни реални стойности, ще наричаме мрежа ако във *V* съществува точно един връх *s,* от който само излизат ребра и точно един връх *t,* в който само влизат ребра. Стойността *cap(i,j)* се нарича капацитет на реброто *(i,j)* (т.е. максималната стойност на потока, който може да “тече” по реброто *(i,j)*), *cost*(*i*,*j*) се нарича цена на реброто *(i,j)* (т.е. колко струва преминаване на единица поток през реброто), върхът *s* – източник, а върхът *t* – цел на мрежата.

Поток с цена в мрежата наричаме функция *f* с реални стойности, дефинирани върху ребрата на мрежата, за която са в сила:

*f(i,j)* ≤ *cap(i,j)* и *f(i,j)* =-*f(j,i)* за всяка двойка върхове *i* и *j*.

за всеки връх *i*, различен от *s* и *t*, сумата от *f(i,j)* за всяко *j* от *V* е равна на 0.

*cost*(*i*,*j*)= -*cost*(*j*,*i*), за всяка двойка върхове *i* и *j*

Стойността на потока наричаме числото |*f*| = ∑j∈V *f(s,j)* = ∑i∈V *f(i,t)*, а цена на потока – *cost*(*f* ) = ∑*f*(*v*,*w*)>0 *cost*(*v*,*w*) *f*(*v*,*w*) = ∑i,j∈V *cost*(*v*,*w*) *f*(*v*,*w*) / 2. Поток има минимална цена, ако не съществува поток *g*, за който |*g*|=|*f*| и cost(*g*) < *cost*(*f*).

В нашата задача е известен потокът, който се търси, но това не пречи задачата да се разглежда като задача за максимален поток. Добавянето на супер-източник и ребро от него към източника със стойност зададения поток свежда задачата до намиране на максимален поток с минимална цена. За да е изпълнено условието *cost*(*i*,*j*)= -*cost*(*j*,*i*), то може да удвоим върховете в графа. Всеки връх *v* се замества с два върха *v’* и *v’’*, свързани с ребро . Всички ребра влизащи във *v* сега влизат във *v’*, а всички излизащи от *v* – излизат от *v’’*. Капацитетът на новото ребро (*v’,* *v’’)* ще е неограничено (т.е. достатъчно голямо число), а цената му 0. Такава конструкция гарантира единствено ребро между всеки два върха на графа след като неориентираните ребра се заменят с две ориентирани (по една във всяка посока). Ребрата чиито край е източника или целта трябва да се ориентират еднопосочно, за да се изпълнени условията за графа.

Остатъчната функция се дефинира като *res*(*i*,*j*) =*cap*(*i*,*j*) - *f*(*i*,*j*), а остатъчната мрежа *R* е граф с върхове върховете *G*, същите източник и цел, ребра двойките (*i*,*j*), за които *res*(*i*,*j*)>0, с капацитет *res*(*i*,*j*) и същата ценова функция.

Решаването на задачата максимален поток с минимална цена има различни подходи. Ето няколко от тях.

Идеята за намаляване на цените се основава на това, че поток има минимална цена тогава и само тогава, когато остатъчната мрежа няма отрицателни цикли. За да намерим максималния поток с минимална цена можем първо да намерим какъвто и да е максимален поток, а после ще намаляваме цената му „пускайки” колкото се може повече поток по отрицателните цикли. Това се повтаря докато има отрицателни цикли. За капацитети/цени, които са реални числа този алгоритъм може никога да не завърши, но при целочислени рано или късно ще приключи. Времето за изпълнение на този алгоритъм не е полиномиално.

Алгоритъмът на Форд-Фълкерсън за намиране на максимален поток използва нарастващи пътища (път от *s* до *t* с ребра с положителен капацитет в остатъчния граф). Така втората идея за намиране на максимален поток с минимална цена използва верността на твърдението, че ако *f* е поток с минимална цена и *p* е нарастващ път с минимална цена, то добавянето на *p* към *f* дава поток с минимална цена. Сложността на този алгоритъм е *O*(<брой стъпки> \* <сложността за намиране нарастващ път с минимална цена>). Ако се използва търсене в ширина за строенето на нарастващия път, то това е O(*N*\**M*\*|*f\**|), където *f\** е максималният поток. Сложността може да бъде подобрена, като се използва трансформация върху цената на ребрата, за да ги направим неотрицателни и да можем да използваме алгоритъма на Дийкстра. Тази трансформация е: *cost*"(*i*,*j*) = *cost*(*i*,*j*) + *dist*(*i*) - *dist*(*j*), където dist(i) e дължината на най-късия път от *s* do *i*. Понеже на всяка стъпка остатъчната мрежа се променя, то трябва да се променят и трансформираните цени. Така сложността може да се сведе до *O*(*N*\**M*+|*f\**|\*(*M*+*N*\*log*N*)).

Използвайки „скалиране” може да се подобри алгоритъмът, когато потокът е много голям. Идея на скалирането при намирането на максимален поток е във всяка стъпка на алгоритъма добавяме поток само по онези ребра в остатъчната мрежа с капацитет ≥ z. Между всеки две стъпки z се намалява наполовина. Тази идея е приложима и при намирането на максимален поток с минимална цена, но чрез някои модификации. Както и по-горе, за по-ефективното реализиране на алгоритъма се изисква трансформация, при която ребрата в остатъчната мрежа не са отрицателни. Така може да се получи алгоритъм със сложност *O*((*N*\**М*+*N*\*log*N*)\*log*C*), който очевидно зависи от броя върхове, броя ребра и максималния капацитет на всяка ребро, но не и от стойността на потока. За по-задълбочено запознаване с тази и други интересни задачи препоръчваме книгата „Network Flows: Theory, Algorithms, and Applications” на Orlin, Magnanti, Ahuja.

Победител в кръга стана Ростислав Руменов от Шумен.