1. Алгоритъм на Форд-Белман

Имаме ориентиран граф G(V, E). Ще работим с матрица на теглата А[i] [j].

Алгоритъма въвежда масив D[] в който ще бъдат записани минималния път път от началния връх до всеки друг връх в графа.

2.1. В началото D[i] = A[s][i] за всяко i от върховете.

2.2. Оптимизираме стойността на D[i], за всяко i :

За всяко j, ако D[i] > D[j] + A[j][i] => D[i] = D[j] + A[j][i]

2.3. Повтаря се стъпка 2 (n-2) пъти.

Сложността на този алгоритъм е O(n^3) , тъй като стъпка две има сложност O(n^2).

Този алгоритъм е валиден за графи с произволни тегла на ребрата. Полезно свойство на тои алгоритъм е, че позволява да се открие отрицателен цикъл. Това лесно се разбира, ако след изпълнение на алгоритъма в имаме D[i] > D[j] + A[j][i].

1. Aлгоритъм на Флойд

Подобен е на този а Форд-Белман. Съществената разлика е, че след като завърши работа, пазполагаме с минималните дължини между всеки сва върха от графа о то без да се заделя допълнително памет (използва се директно мареицата на теглата). Това означава, че в края стойността на A[i] [j] ще бъде минималния път между i и j.

Работата на алгоритъма се състои в следното: за всеки два върха i, j на A[i][j] се присвоява по-малкото от **А[i] [j] и A[i][k] + A[k][i]**, за всеки връх k. Върхът k се използва за оптимизиране на пътя.

Сложността на този алгоритъм е кубична O(n^3). Наличието на отрицателни ребра не представлява проблем за алгоритъма.

За реализация на алгоритъма може да се използват три цикъла.

При сравняване на основната стъпка ще използваме някаква максимална стойност с която да означаваме липсата на на ребро в матрицата. За да работи коректно трябва MAX\_VALUE да има стойност по-голяма от максималното тегло на ребро в графа.

Алгоритъм на Дейкстра

Условието за да работи и да не зацикли е всички ребра да имат положителни стойности. Най-общо казано, за да се намери най-кратък път между два върха трябва да се намери най краткия път сред междунните им. w(s, i) = w(s, j) + w(j, i), където w – най-къс път.

Стъпки на работа:

1. Инициализираме масив d[ ] като:

* d[ i ] = graph[s, i], за всеки съсед i на s
* d[ i ] = MAX\_VALUE, за всеки връх i, несъсед на s

Въвеждаме множество T, което в началото съдържа всички върхове на графа, без s (нашия сорс)

1. Докато T съдържа поне един връх i, за който d[ i ] < MAX\_VALUE:

* Избираме връх j от T такъв, че d[ j ] да бъде минимално.
* Изключваме j от T
* За всяко i от T изпълняваме d[ i ] = min(d[ i ], d[ j ] + A[ j, i ])

1. За да възстановим върховете , участващи в минимален път (а не само дължината на пътя), ще въседем допълнителен масив pred[ ]. Така в i-тата позиция на масиваа pred[ ] се записва връх j, за който w(s, j) + f(j, i) е било минимално.
2. **Алгоритъмът на Floyd Warshall се използва на намиране на най-преките пътища от връх до връх в претеглен граф.Той е със сложност n^3 и при него както при Ford могат да се намерят отрицателни цикли и съответно да може да се каже дали алгоритъмът може да се използва за даден претеглен граф.Сравнявайки го с дийкстра откриваме ,че използвайки дийкстра от всеки до всеки връх получаваме сложност O(VE + V^2lg(V)) ,която съответно показва ,че Floyd Warshall е по-бързият в конкретната ситуация.Други + -ове на алгоритъмът са работата с отрицателни пътища,сравнително лесна имплементация.**
3. Рекурсивен алг.е този,който при изпълн.си се обръща към себе си пряко или косвено поне 1 път.За да реализираме рекурс.алг. използ.*рекурс.ф-ии*. Има 2 вида рекурсия:
4. - **проста** – в тялото си алг.се обръща поне 1 път към себе си пряко,но с др.аргументи, които постеп.намал.и се стига до кр.ст-сти, при които алг.не се обръща към себе си
5. **- взаимна** – алг.А1 се обръща на 1 или по-вече места към А2 и обратно. Мд има м/у повече от 2 алг.

Дърветата са математически абстракции, които играят важна роля при конструирането и анализа на алгоритми. От една страна използваме дърветата, за да изучим свойствата на алгоритмите, а от друга ги използваме като ясно формулирани структури от данни.

**Def.** **Дърво е не празна съвкупност от върхове и ребра**.

**Връх** (възел) е прост обект, които може да има име и да носи друга асоциативна информация.

**Ребро** връзка между два върха.

**Път** е списък от различни върхове, в които посл. върхове са свързани с ребра в дървото.

Дефиниращо свойство за дърво е, че има само един път свързващ всеки два възела.

Възлите директно под даден възел се наричат негови **деца.**

Възлите без деца се наричат **листа.**

**Корен на дървото** е възел, от които произлизат всички останали възли.

**Видове дървета:**

**Бинарни дървета** представлява наредено дърво, състоящо се от два типа възли външни, които нямат деца и вътрешни, които имат точно две деца.

**Наредени дървета-** **дърво с корен,** в които е определен реда на децата за всеки възел.

**Дърво с корен** е това при което един възел е определен за корен на дървото.

Бинарно дърво е или външен възел или вътрешен възел, свързан към двойка бинарни дървета, която се нарича ляво или дясно дърво на този възел.

**Математически свойства на бинарните дървета**

**1св-во:** Бинарно дърво с n на брой вътрешни възли има n+1 на брой външни възела.

**2св-во:** Бинарно дърво с n на брой вътрешни възли има 2 n на брой връзки. (n-1 на брой връзки към вътрешни възли и n+1 на брой връзки към вътрешни възли.

**3св-во:** Нивото на даден възел в дърво е по голямо с едно от нивото на неговия родител.

**4св-во:** Височината на бинарно дърво е максимума от нивата на неговите възли.

**5св-во:** Дължината на пътищата в бинарно дърво е сумата от нивата на неговите възли.

Списък е линейна структура от данни, която съдържа поредица от елементи. Списъкът има свойството дължина (брой елементи) и елемен¬тите му са наредени последователно.

Списъкът позволява добавяне на елементи на всяко едно място, премахването им и последователното им обхождането. Както споменахме по-горе, един АТД може да има няколко реализации.

Статичен списък (реализация чрез масив)

Масивите изпълняват много от условията на АТД списък, но имат една съществена разлика – списъците позволяват добавяне на нови елементи, докато масивите имат фиксиран размер.

Въпреки това е възможна реализация на списък чрез масив, който автоматично увеличава размера си при нужда . Такъв списък се нарича статичен.

Свързан списък (динамична реализация)

Както видяхме, статичният списък има един сериозен недостатък – опера¬циите добавяне и премахване от вътрешността на списъка изискват пренареждане на елементите. При често добавяне и премахване (особено при голям брой елементи) това може да доведе до ниска производителност. В такива случаи се използват т. нар. свързани списъци. Разликата при тях е в структурата на елементите – докато при статичния списък елементите съдържат само конкретния обект, при динамичния списък елементите пазят информация за следващия елемент. Ето как изглежда един примерен свързан списък в паметта:

**При хеш-таблиците, ако разполагаме с ключ, броят сравнения, които трябва да извършим, за да установим има ли стойност с такъв ключ, е константен и не зависи от броя на елементите в нея. Как точно се постига такава ефективност ще разгледаме в детайли по-долу.**

**Когато реализациите на някои структури от данни ни дават време за достъп до елементите й, независещ от броя на елементите в нея, се казва, че те притежават свойството random access (свободен достъп). Такова свойство обикновено се наблюдава при реализации на абстрактни струк-тури от данни с хеш-таблици и масиви.**

**Какво е хеш-таблица?**

**Структурата от данни хеш-таблица обикновено се реализира с масив. Тя съдържа наредени двойки (ключ, стойност), които са разположени в масива на пръв поглед случайно и непоследователно. В позициите, в които нямаме наредена двойка, имаме празен елемент (null):**

**Краен ориентиран граф се нарича наредената двойка G=(V,E), където:**

**V={v1,v2,…,vn} е крайно множ.от върхове;**

**E={e1,e2,…,em} е крайно множ.от ориентирани ребра. Всеки елемент (k=1,2,…,m) е наредена двойка (vi,vj), , 1≤i,j≤n.**

**Ако ребрата на графа са неориентирани, графът се нарича неориентиран.**

**Върхът w е съседен на v тогава и само тогава, когато в неориентиран граф, ако w е съседен на v, то и v е съседен на w.**

**Път в граф ще наричаме последователността от върхове v1,v2,v3,…,vn, такава че , 1≤i≤n. Дължина на пътя се нарича броя на ребрата в него. Ще допускаме път от върха, до самия него, като ако няма ребра, дължината му е нула. Ако графът съдържа ребро (v,v), пътят v,v се нарича примка. Ще считаме, че графът по подразбиране няма примки.Прост път се нарича път, в който всички върхове са различни, с изключение на първия и последния.**