

KRUSKAL GRAPHS

HELP

MINIMUM SPANNING TREE'S

BEHZAD KHOSRAVIFAR





Introduction



Kruskal Algorithm Structure



Kruskal Graph's Workmanship



Example For Work By
Kruskal Graph's Program

INTRODUCTION

Tab 1

مقدمه



برنامه Kruskal Graph's برگرفته از الگوریتم Kruskal است که از جمله الگوریتم های Greedy یا همان حریصانه به شمار می رود که روشی برای پیدا کردن کوچکترین درخت پوشا (Minimum Spanning tree) از داخل یک گراف می باشد. البته این الگوریتم می تواند در حالت های مختلف (ترتیب) ورودی، برای یک گراف جواب های مختلفی داشته باشد، ولی حتما هزینه ی الگوریتم، برای هر جواب به دست آمده از یک گراف یکی می باشد، زیرا ممکن است ما چندین راه با طول یکسان برای دو راس داشته باشیم، که همه آن راه ها صحیح باشند. البته برای پیدا کردن درخت فوق الگوریتم های مختلفی وجود دارد مانند الگوریتم پریم (Prime)، ولی کم هزینه ترین الگوریتم یافت شده همان الگوریتم کروسکال می باشد. پیچیدگی زمانی الگوریتم $O(n \cdot \log n)$ می باشد، که در آن m تعداد یال ها و n تعداد رئوس گراف G است.

این نوع الگوریتم ها دارای کاربردهای مختلفی هستند، که از جمله آنها می توان به کاربردهای زیر اشاره کرد:

- پیدا کردن کوتاهترین مسیر بین دو نقطه
- شبکه AOV برای کنترل پروژه و فعالیت ها و ارتباط بین آنها
- مسیر بحرانی

امیدواریم از استفاده این برنامه بهره کامل را برده باشید.

KRUSKAL ALGORITHM STRUCTURE

ساختار الگوریتم کروسکال

Tab 2

یک درخت پوشا (Spanning Tree) زیر گرافی از گراف G است که شامل کلیه رئوس گراف G باشد و یک درخت باشد. بنابراین اگر گراف G دارای n گره باشد درخت پوشای آن دارای $n-1$ یال است. تعداد درخت های پوشای یک گراف کامل با n گره برابر $2^{n-1} - 1$ است.

پیمایش های DFS (جستجوی اول عمق Deep First Search) و BFS (جستجوی اول سطح Breadth First Search) هر کدام یک درخت پوشا تولید می کنند.

درخت پوشای حداقل (Minimum Spanning Tree) گراف وزن دار G ، درخت پوشائی است که مجموع وزن های آن حداقل باشد. برای بدست آوردن درخت پوشای هم در اینجا از الگوریتم کروسکال (Kruskal) استفاده می کنیم.

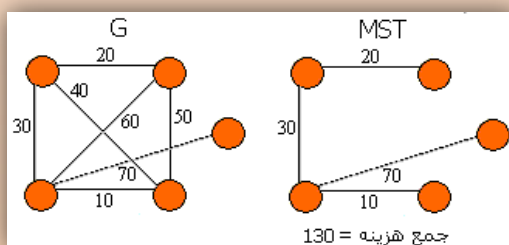
الگوریتم کروسکال

گراف G با n راس را در نظر بگیرید. الگوریتم کروسکال به صورت زیر عمل می کند:

1. تمام یال ها را به طور صعودی بر حسب وزن مرتب کنید.
2. درخت T را متشکل از گره های G بدون یال را ایجاد کنید.
3. عملیات زیر را $n-1$ با تکرار کنید:

یک یال با حداقل وزن را به درخت T اضافه کنید به طوریکه حلقه ایجاد نشود.

گاهی چند یال دارای یک وزن هستند، در این حالت ترتیب یال هایی که انتخاب می شوند مهم نیست. درخت های پوشای حداقل مختلفی ممکن است حاصل شود اما مجموع وزن آنها همیشه یکسان و حداقل می شود.



الگوریتم کروسکال

Input (form user)

گرفتن (مقادیر از کاربر)

1) Get number of vertexes (stored in an integer variable)

1. تعداد رئوس را می گیریم (در متغیر عددی ذخیره می کنیم)

2) Get edges information from user one by one and stored in linked list. The edge information consist of

- * Weight for each edge
- * starting and ending vertex

2. اطلاعات مربوط به یال ها را از کاربر گرفته و در لیست پیوندی ذخیره کنید. اطلاعات یال ها شامل :

- * وزن (طول) هر یال
- * راس مبدا و مقصد یال

Notice that the edges are represented by nodes (see code snippet for data structure code in use)

توجه به اینکه رئوس نماینده و نشان دهنده ی یال ها هستند. (در زیر ریز کدی برای ساختار تعریف کنید)

```

1: struct edge;
2: typedef struct Edge *PtrToEdge;
3:
4: struct edge
5: {
6:     int startingVertex;
7:     int endingVertex;
8:     int weight;
9:     PtrToEdge next;
10: };

```

Process:

Find minimum spanning tree using Kruskal's Algorithm

- * Sort linked list into an ascending order based on weight
- * Delete rejected edges (nodes)
- * Stop once number of accepted edges = (number of vertexes - 1)

پردازش:

پیدا کردن کوچکترین درخت پوشا با استفاده از الگوریتم کروسکال

* لیست پیوندی را به صورت صعودی و بر حسب وزن (طول) یال مرتب کنید

* یال های (رئوس) رد شده را حذف کنید

* یکبار تعداد یال های قبول شده را بشمارید تا برابر $= 1 - \text{تعداد رئوس}$

Output:

Display original graph

Display minimum spanning tree

خروجی :

گراف اصلی را نمایش دهید

کوچکترین درخت پوشا را نمایش دهید

Example

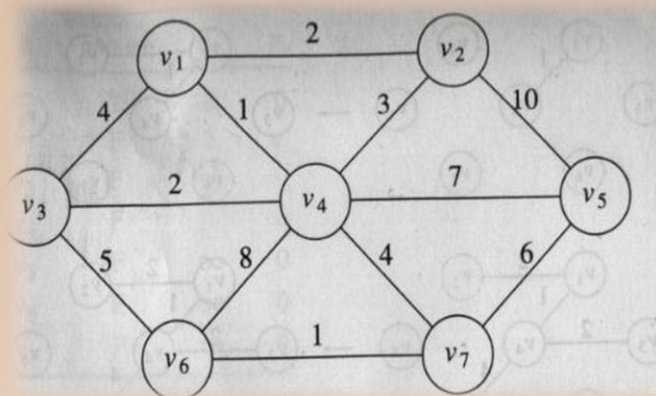
Suppose we have the following graph (Graph in Shape G1)

Using Kruskal's Algorithm

برای مثال

فرض کنید ما یک گراف در زیر داریم (گراف شکل G1)
با استفاده از الگوریتم کروسکال

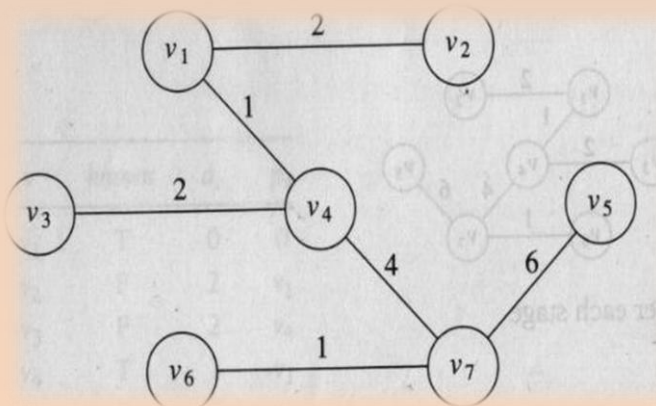
Edge Name	Weight	Action
(v1, v4)	1	Accepted
(v6, v7)	1	Accepted
(v1, v2)	2	Accepted
(v3, v4)	2	Accepted
(v2, v4)	3	Rejected
(v1, v3)	4	Rejected
(v4, v7)	4	Accepted
(v3, v6)	5	Rejected
(v5, v7)	6	Accepted



شکل G1

The resulting minimum spanning tree is shown in Shape G2

نتیجه ی کوچکترین درخت پوشا در شکل G2 نشان داده شده



شکل G2

KRUSKAL GRAPH'S WORKMANSHIP

طرز کار برنامه گرافه کروسکال

برای اینکه بتوانید برنامه را نصب کنید باید ابتدا محیط NET Framework 3.5 SP1 . بر روی کامپیوتر شما نصب شده باشد، که البته این برنامه همراه برنامه ی اصلی می باشد که در صورت عدم وجود آن در کامپیوترتان، به طور خودکار نصب خواهد؛ که مدت زمانی طول خواهد کشید تا فرایند نصب ظاهر شود.

بعد از نصب برنامه و اجرای آن، تعداد یال های گراف اصلی را در تکست باکس سمت چپ بالای صفحه وارد کرده و کلید اینتر Enter را فشار دهید. این برنامه نیازی به تعداد رئوس گراف شما ندارد زیرا خود برنامه می تواند تعداد رئوس را از اطلاعات یال های وارده (راس ابتدا و انتهای یال) بدست آورد.

پس از زدن کلید اینتر Enter به تعداد یال هایی که شما وارد کرده اید، برنامه خانه می سازد تا اطلاعات یالها را با وزن (طول) شان وارد کنید. که این عمل هم بسته به تعداد یال های وارده و سرعت CPU کامپیوتر شما مدت زمانی طول خواهد کشید. لازم به ذکر است که نیازی به مرتب وارد کردن مشخصات بر حسب وزن یال ها ندارد. برای مثال می خواهیم فرایند حل یک گراف را از اولین مرحله تا آخرین مرحله با توجه به الگوریتم زیر حل کنیم :

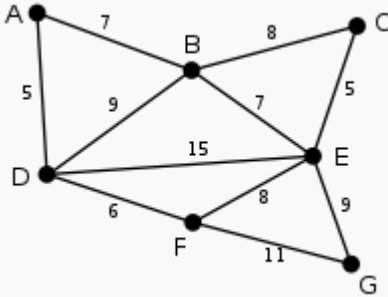
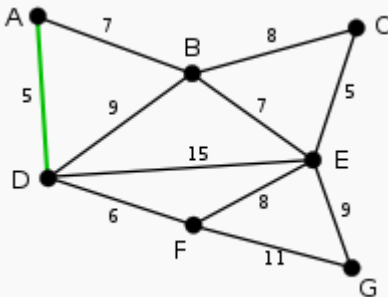
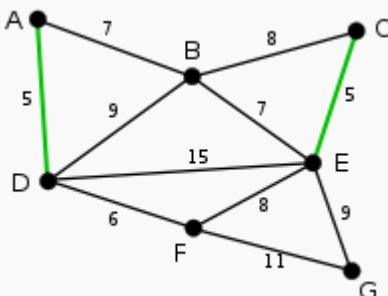
```
1  function Kruskal(G)
2      for each vertex v in G do
3          Define an elementary cluster  $C(v) \leftarrow \{v\}$ .
4      Initialize a priority queue Q to contain all edges in G, using
the weights as keys.
5      Define a tree  $T \leftarrow \emptyset$  //T will ultimately contain the
edges of the MST
6          // n is total number of vertices
7      while T has fewer than n-1 edges do
8          // edge u,v is the minimum weighted route from/to v
9           $(u,v) \leftarrow Q.removeMin()$ 
10         // prevent cycles in T. add u,v only if T does not already
contain a path between u and v.
11         // Note that the cluster contains more than one vertex only if
an edge containing a pair of
12         // the vertices has been added to the tree.
```

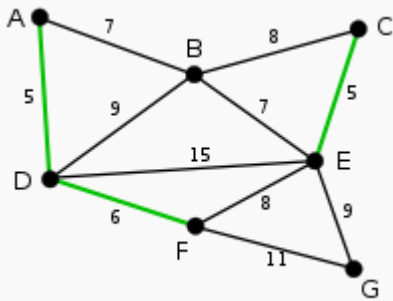
```

13   Let  $C(v)$  be the cluster containing  $v$ , and let  $C(u)$  be the cluster
containing  $u$ .
14   if  $C(v) \neq C(u)$  then
15       Add edge  $(v,u)$  to  $T$ .
16       Merge  $C(v)$  and  $C(u)$  into one cluster, that is, union  $C(v)$  and  $C(u)$ .
17   return tree  $T$ 

```

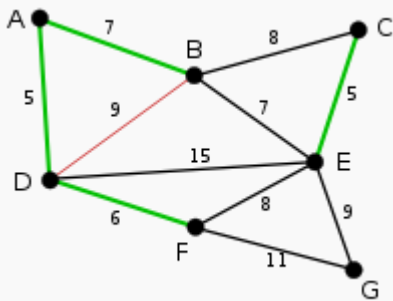
Example

Image	Description
	<p>This is our original graph. The numbers near the arcs indicate their weight. None of the arcs are highlighted.</p> <p>این گراف اصلی ما است. اعداد نزدیک یال ها وزنشان (طولشان) را نشان می دهد. هیچ یک از کمان ها یا همان یال ها پر رنگ نیست (انتخاب نشده است).</p>
	<p>AD and CE are the shortest arcs, with length 5, and AD has been arbitrarily chosen, so it is highlighted.</p> <p>AD و CE با طول 5، کوتاهترین کمان ها هستند؛ و AD اتفاقی انتخاب شده است، بنابراین آن را پر رنگ کردیم.</p>
	<p>CE is now the shortest arc that does not form a cycle, with length 5, so it is highlighted as the second arc.</p> <p>حالا CE کوتاهترین کمان است با طول 5 و یک حلقه (Cycle) تشکیل نمی دهد. بنابراین آن را هم پر رنگ کردیم به عنوان دومین کمان.</p>



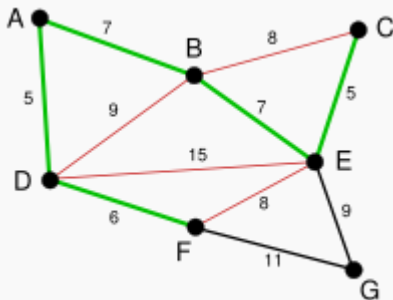
The next arc, **DF** with length 6, is highlighted using much the same method.

کمان بعدی DF با طول 6 را پر رنگ کردیم و تقریباً از همان روش استفاده کردیم.



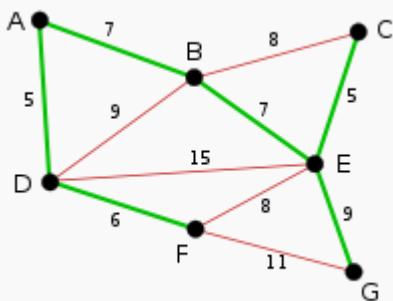
The next-shortest arcs are **AB** and **BE**, both with length 7. **AB** is chosen arbitrarily, and is highlighted. The arc **BD** has been highlighted in red, because there already exists a path (in green) between **B** and **D**, so it would form a cycle (**ABD**) if it were chosen.

بعد کوتاهترین کمان ها AB و BE هستند، هر دو با طول 7. AB به طور اتفاقی انتخاب شده است (چون در اول لیست بود)، و آن را پر رنگ کردیم. کمان BD را با رنگ قرمز پر رنگ کردیم، زیرا بین B و D مسیر (سبز) وجود داشت، بنابراین اگر آن را انتخاب می کردیم یک حلقه (**ABD**) تشکیل می شد.



The process continues to highlight the next-smallest arc, **BE** with length 7. Many more arcs are highlighted in red at this stage: **BC** because it would form the loop **BCE**, **DE** because it would form the loop **DEBA**, and **FE** because it would form **FEBAD**.

فرآیند به پر رنگ کردن کوتاهترین کمان بعدی ادامه می دهد، BE با طول 7. در این مرحله کمان های زیادی با رنگ قرمز پر رنگ می شوند. مثل BC زیرا حلقه BCE را تشکیل می دهد و DE زیرا حلقه DEBA را تشکیل میدهد و FE زیرا حلقه FEBAD را تشکیل می دهد.



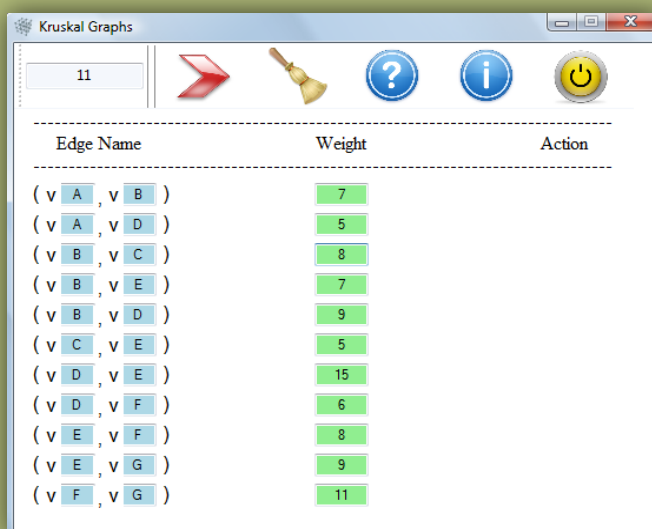
Finally, the process finishes with the arc **EG** of length 9, and the minimum spanning tree is found.

بالاخره، فرآیند با کمان EG با طول 9 به پایان می رسد، و کوچکترین درخت پوشا پیدا می شود.

EXAMPLE FOR WORK BY KRUSKAL GRAPH'S PROGRAM

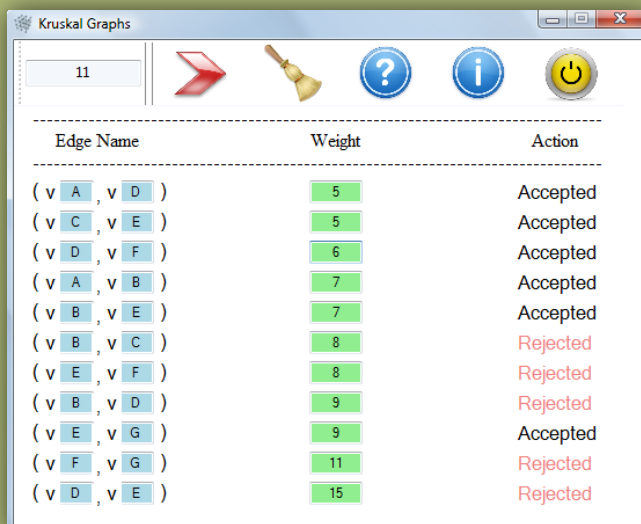
مثالی برای کار با برنامه گرافه کروسکال

در اینجا فرآیند کار با برنامه را با مثال Tab 3 که به صورت نمودارهای گرافیکی بیان شده بود را اجرا میکنیم.
گراف مورد نظر ما 11 یال داشت و 7 راس $\text{Vertex} = \{A, B, C, D, E, F, G\}$



The screenshot shows the 'Kruskal Graphs' application window. At the top, there is a toolbar with icons for a red arrow, a bell, a question mark, an information icon, and a power button. Below the toolbar is a table with three columns: 'Edge Name', 'Weight', and 'Action'. The table lists 11 edges with their respective weights.

Edge Name	Weight	Action
(v A , v B)	7	
(v A , v D)	5	
(v B , v C)	8	
(v B , v E)	7	
(v B , v D)	9	
(v C , v E)	5	
(v D , v E)	15	
(v D , v F)	6	
(v E , v F)	8	
(v E , v G)	9	
(v F , v G)	11	



The screenshot shows the 'Kruskal Graphs' application window after running the algorithm. The table now includes an 'Action' column, indicating whether each edge was 'Accepted' or 'Rejected'.

Edge Name	Weight	Action
(v A , v D)	5	Accepted
(v C , v E)	5	Accepted
(v D , v F)	6	Accepted
(v A , v B)	7	Accepted
(v B , v E)	7	Accepted
(v B , v C)	8	Rejected
(v E , v F)	8	Rejected
(v B , v D)	9	Rejected
(v E , v G)	9	Accepted
(v F , v G)	11	Rejected
(v D , v E)	15	Rejected