# MỞ RỘNG THUẬT TOÁN DIJKSTRA

**Vũ Đình Hòa, Tạ Anh Sơn, Đỗ Thị Bích Ngọc**
Khoa Công nghệ Thông tin, Đại học Sư phạm Hà nội
*(Bài nhận ngày 02 tháng 06 năm 2005, hoàn chỉnh sửa chữa ngày 11 tháng 08 năm 2005)*

***TÓM TẮT:*** *Vấn đề đường đi ngắn nhất là một vấn đề hết sức quan trọng trong lý thuyết luồng trên mạng. Chúng xuất hiện thường xuyên trong thực tế và có những ứng dụng hết sức phong phú khi chúng ta muốn gửi một vài tài liệu (như các gói dữ liệu, một cuộc điện thoại, như là một phương tiện vận chuyển) giữa hai nút nào đó trong mạng một cách nhanh nhất, rẻ nhất, thuận tiện nhất có thể,.... Đã có nhiều tác giả quan tâm đến vấn đề này va 2đã có nhiều kết quả được công bố. Một thuật toán được đưa ra gởi Dijsktra (1959) cho đến nay vẫn được sử dụng rộng rãi do tính đơn giản và hiệu quả của nó, .... Trong bài viết này chúng ta sẽ đưa ra một cách sử dụng thuật toán Dijsktra nhằm giúp chúng ta dễ sử dụng và trực quan hơn. Hơn nữa trong thuật toán Dijsktra cổ điển chúng ta phải gắn lại nhãn cho các đĩnh trong mỗi bước, nên trong bài viết này chúng ta sử dụng các mũi tên gắn cho mỗi đỉnh nhãn có đúng một lần mà không cần thiết khởi tạo các giá trị nhãn* $\infty$.

Trong phần này chúng ta sẽ giới thiệu thuật toán Dijkstra cổ điển, một số cách áp dụng khác nhau và một số các dạng khác của thuật toán Dijkstra. Thuật toán Dijkstra chỉ xét các đồ thị có hướng G=(V,E), |V|=n, |E|=m với các cung được gán trọng số không âm, nghĩa là, mỗi cung (u, v)$\in$E của nó được đặt tương ứng với một số thực c(u,v) gọi là trọng số của nó. Chúng ta sẽ đặt c(u, v) = $\infty$, nếu (u, v)$\notin$E. Nếu dãy $v_0$, $v_1$,... $v_p$ là một, đường đi trên G, thì độ dài của nó được định nghĩa là tổng sau: $\sum_{i=0}^{p-1} c(v_i, v_{i+1})$ tức là, độ dài của đường đi chính là tổng các trọng số trên các cung của nó.

Bài toán tìm đường đi ngắn nhất trên đồ thị dưới dạng tổng quát có thể phát biểu như sau: tìm đường đi có độ dài nhỏ nhất từ một đỉnh xuất phát từ s$\in$V đến đỉnh cuối t$\in$V. Đường đi như vậy được gọi là đường đi ngắn nhất từ s đến t, còn độ dài của nó ta sẽ ký hiệu là d(s, t) và còn được gọi là khoảng cách từ s đến t. Trong trường hợp không tồn tại đường đi từ s đến t, ta đặt d(s, t)= $\infty$. Rõ ràng, đường đi ngắn nhất không có đỉnh lặp lại.

## 1. THUẬT TOÁN DIJSKTRA CỔ ĐIỂN

Thuật toán Dijkstra là thuật toán tìm đường đi ngắn nhất từ một nút nguồn s tới mọi nút khác trong mạng với độ dài các cung là không âm. Thuật toán gán các nhãn khoảng cách d(i) cho mỗi nút i' với d(i) là cận trên cho đường đi ngắn nhất tới nút i. Tại mỗi bước thuật toán chia các nút vào hai nhóm: một nhóm **cố định** các nhãn biểu diễn đường đi ngắn nhất và còn lại vào nhóm **thay đổi** các nhãn là cận trên cho đường đi ngắn nhất và sẽ được biến đổi trong các bước tiến hành thuật toán sau đó. Khởi tạo chúng ta gán cho nút s **cố định** nhãn bằng 0 các nút còn lại thuộc nhóm **thay đổi** nhãn bằng$\infty$. Thuật toán sẽ chon một nút i với nhãn thay đổi nhỏ nhất đưa vào nhóm cố định và xoá nó khỏi nhóm thay đổi sau đó duyệt các cung thuộc A(i) để cập nhật nhãn của các nút kề với nó. Khi nào tất cả các nhãn được gán cố định thì thuật toán kết thúc. (Với A(i) là tập các cung đi ra từ i.)
Thuật toán kết thúc cho chúng ta một cây *T* với gốc là nút s với các đỉnh là các nút có nhãn hữu hạn. Với mỗi (i,j)$\in$*T* ta có pred(j)=i và d(j)=d(i)+$c_{ij}$.

**Thuật toán Dijkstra**

begin

S:=$\phi$; $\overline{S}$:=V;

d(i):=$\infty$ với mỗi i$\in$E;

d(s):=0; pred(s):=0;

while |S|< n do

begin

i $\in \overline{S}$ là nút sao cho d(i)= min {d(j): j$\in \overline{S}$ };

S:=S $\cup$ {i};

$\overline{S}$:= $\overline{S}$-{i};

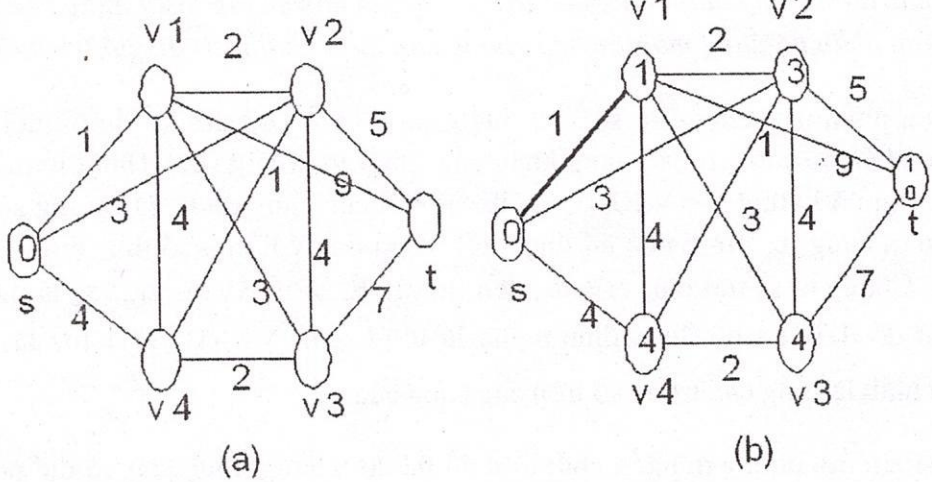
for (i,j) $\in$ A(i) do

if d(j) > d(i)+ $c_{ij}$ then d(j):=d(i)+$c_{ij}$ and pred(j):=i;

end;

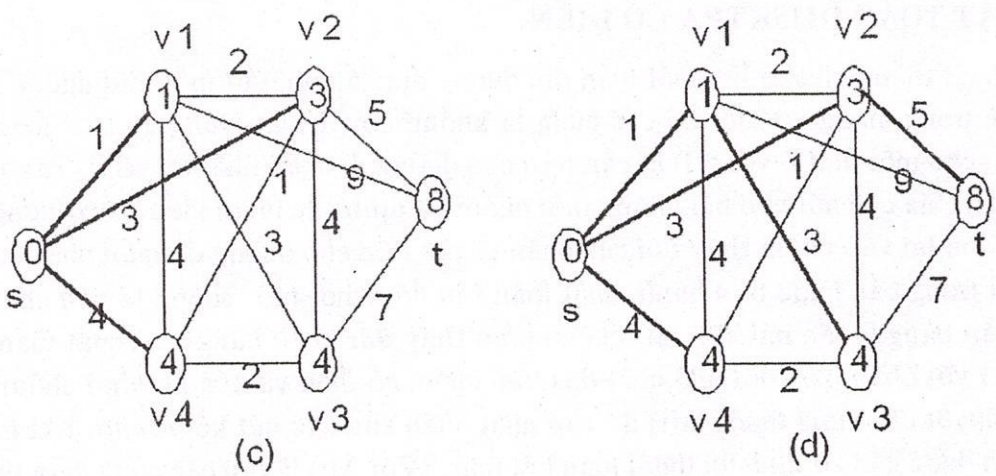
end;

Chúng ta xét ví dụ minh họa sau.




**Hình 1**

Với đồ thị và các trọng số cho ở hình (a) với yêu cầu tìm đường đi ngắn nhất từ s tới t. Bước đầu tiên gán nhãn của s là 0 và gán nhãn cho tất cả các đỉnh còn lại là $\infty$.




**Hình 2**

Bước tiếp theo chúng ta đưa đỉnh s vào tập nhãn cố định và sửa nhãn các đỉnh $v_1,v_2,v_4$ là 1,3,4 từ đó chúng ta lựa chọn đỉnh có nhãn nhỏ nhất $v_1$ đưa vào tập S. Sau đó ta tính các nhãn mới được các nhãn của $v_3$ là 4 còn các nhãn khác không thay đổi, chúng ta so sánh các nhãn của các đỉnh có nhãn thay đổi để tìm đỉnh có nhãn nhỏ nhất ta được $v_2$ và đưa vào đỉnh có nhãn cố định. Tương tự các bước tiếp theo chúng ta lần lượt đưa được các đỉnh $v_4$, $v_3$, t và cuối cùng thu được nhãn của t là 8 sau 5 bước và nhiều lần sửa nhãn.

**Tính đúng đắn của thuật toán:** Chúng ta có thể dễ dàng chứng minh được tính đúng đắn của thuật toán bằng quy nạp theo số phần tử của S.

**Thời gian tính toán của thuật toán Dijkstra:** Chúng ta sẽ tìm hiểu xem trường hợp xấu nhất của thuật toán Dijkstra xảy ra như thế nào. Chúng ta thấy rằng thời gian tính toán của thuật toán Dijkstra chủ yếu gồm hai quá trình cơ bản sau:

- **Lựa chọn nút:** thuật toán thực hiện n lần mỗi lần đòi hỏi phải duyệt nhóm nút có nhãn thay đổi. Do đó chúng ta có tổng thời gian tính của lựa chọn nút là: $n+(n-1)+ \dots +1 = O(n^2)$.
- **nhật khoảng cách:** Thuật toán sẽ thực hiện quá trình này trong $|A(i)|$ lần với mỗi nút i. Ta có $\sum_{i\in V} |A(i)|= m$ và mỗi lần cập nhật sẽ mất O(1) lần, như vậythuật toán đòi hỏi O(m) thời gian cho toàn bộ quá trình cập nhật khoảng cách.

Từ đó chúng ta thấy thời gian thực hiện thuật toán là $O(n^2+m)=O(n^2)$.

### 1.1. Thuật toán Dijkstra ngược

Thay vì yêu cầu xác định đường đi ngắn nhất từ một nút nguồn s tới các nút khác chúng ta muốn xác định đường đi ngắn nhất từ các nút tới một nút đáy t. Để giải quyết vấn đề chúng ta thấy ngay đây là bài toán biến đổi từ thuật toán Dijstra chúng ta gọi là thuật toán Dijstra ngược. Thuật toán này giữ lại các khoảng cách d'(j) với mỗi nút j là cận trên cho đường đi ngắn nhất từ nút j tới nút t. Như trước chúng ta thiết kế một tập S' như là tập nhãn **cố định** phần còn lại là $\overline{S'}$ được gọi là tập nhãn thay đổi. Tại mỗi bước của thuật toán chúng ta chọn một nút với nhãn khoảng cách
thay đổi là nhỏ nhất, gọi là d'(j) và đưa vào cố định. Sau đó chúng ta kiểm tra các cung tới (i,j) và thay đổi nhãn của i bởi $\min\{d'(i), c_{ij}+d'(j)\}$. Thuật toán kết thúc khi mọi định được đưa vào tập cố định.

### 1.2. Thuật toán Dijkstra từ hai phía

Trong một vài ứng dụng của vấn đề đường đi ngắn nhất chúng ta không cần xác định đường đi ngắn nhất từ nút s tới mọi nút trong mạng. Giả sử rằng chúng ta cần xác định đường đi ngắn nhất từ nút s tới nút t nào đó khi đó chúng ta có thể sử dụng thuật toán Dijkstra và dừng thuật toán khi nút t được chọn vào tập S cho dù vẫn còn các đỉnh thuộc tập $\overline{S}$ chưa được xét tới. Thuật toán Dijkstra từ hai phía mà chúng ta mô tả sau đây trong nhiều trường hợp cho chúng ta kết quả nhanh hơn. Trong thuật toán Dijkstra từ hai phía chúng ta áp dụng đồng thời thuật toán Dijkstra từ nút s và thuật toán Dijkstra ngược từ nút t. Thuật toán thiết kế đưa một nút trong $\overline{S}$ và một nút trong $\overline{S'}$ vào tập cố định (tương ứng) cho đến khi cả thuật toán thuận và nghịch có cùng nhãn cố định tại một nút gọi là nút k khi đó ta có đường đi ngắn nhất là $P(k)\cup P'(k)$.

### 1.3. Dial's implementation

Chúng ta nhận thấy rằng nhãn khoảng cách theo thuật toán Dijkstra trong tập cố định là không giảm do độ dài các cung là không âm. Thuật toán dial sắp xếp các nhãn biến đổi

trong các tập phù hợp. Chúng ta lấy nC+1 tập, gán nhãn cho các tập tương ứng lần lượt là 0,1,2,...,nC. Tập có nhãn k chứa các nút có nhãn khoảng cách thay đổi bằng k. Với C là độ dài lớn nhất của cung trong mạng và do đó nC là cận trên cho nhãn khoảng cách của mọi nhãn hữu hạn trong mạng. Chúng ta biểu diễn lại các nút trong tập nhãn k bởi tập content(k). Quá trình lựa chọn nút được thực hiện bằng cách kiểm tra các gói 0,1,2... cho tới khi xác định được gói đầu tiên khác rỗng. Giả sử rằng gói đầu tiên là k. Với mỗi nút trong content(k) có nhãn khoảng cách nhỏ nhất. Lần lượt chúng ta xoá các nút khỏi gói, gán cho chúng các nhãn cố định và duyệt các cung kề để cập nhật các nhãn khoảng cách của các nút kề. Ta có thể dễ dàng chỉ ra rằng độ phức tạp của thuật toán là O(m+nC).

## 2. CÁC ĐỀ XUẤT CẢI TIẾN THUẬT TOÁN

Các thuật toán Dijkstra đề cập ở trên đều phải thực hiện gán nhãn cho tất cả các đỉnh và phải thực hiện kiểm tra lại nhãn và sửa nhãn nhiều lần. Để minh hoạ chúng ta cũng phải vẽ nhiều hình để tránh bị mất các nhãn đã gán...Chúng tôi đề xuất một cải tiến để khắc phục các yếu điểm trên bằng cách sử dụng mũi tên hướng và chỉ thực hiện tính toán với cung có trọng số nhỏ nhất hiện thời.

Trước hết ta chỉ cần khảo sát các đồ thị mà trọng số các cạnh là số tự nhiên. Việc gán giá trị tự nhiên cho các trọng số không giảm ý nghĩa của lời giải khi chúng ta sử dụng công cụ máy tính để xác định con đường ngắn nhất từ đỉnh nguồn s tới đỉnh đích t. Bài toán tìm đường đi ngắn nhất từ đỉnh nguồn s tới đỉnh đích t được phát biểu một cách tổng quát: Cho trước đồ thị có trọng số tự nhiên G với một đỉnh nguồn s. Tìm đường đi ngắn nhất từ đỉnh nguồn s tới tất cả các đỉnh còn lại của đồ thị. Ta có thể đưa ra một thuật toán như sau:

1. *Lấy k=0 và gán cho đỉnh s nhãn 0.*
2. *k:=k+1 và xác định đỉnh x của đồ thị chưa có nhãn sao cho tồn tại một con đường có độ dài k nối s với x.*
3. *Thuật toán kết thúc khi đỉnh t có nhãn (nhãn của t chính là độ dài con đường ngắn nhất nối s với t hoặc khi k > tổng trọng số các cạnh và t chưa có nhãn (khi đó không tồn tại con đường nối s với t).*

Như đã biết, thuật toán khi thực hiện sẽ phải xét lần lượt từ đỉnh nhãn bằng 0, sau đó phải thực hiện lần lượt kiểm tra lại nhãn nhiều lần mỗi lần tăng lên một đơn vị. Nhưng thực ra chúng ta chỉ cần xét giá trị nhãn nhỏ nhất trong các nhãn sau mỗi lần ta kết nạp thêm một đỉnh mới. Để khắc phục điều này, chúng tôi đề xuất sử dụng thêm các mũi tên hướng cho từng đỉnh như sau: Tại đỉnh hiện thời, ta gán các mũi tên theo các cung xuất phát từ đỉnh đó tới các đỉnh chưa được gán nhãn, các mũi tên này có trọng số bằng tổng nhãn của đỉnh đó với trọng số của cung tương ứng. Trong tất cả các mũi tên trên đồ thị, chọn ra mũi tên có trọng số nhỏ nhất. Ta thực hiện gán cho đỉnh đến của cung tương ứng với mũi tên nhãn bằng trọng số của mũi tên, sau đó, thực hiện loại tất cả các mũi tên đến đỉnh đó khỏi đồ thị và coi đỉnh đó là đỉnh hiện thời. Thực hiện lại các bước trên. Nếu trong quá trình thực hiện, đỉnh đích được gán nhãn thì nhãn đó chính là độ dài đường đi ngắn nhất. Trong trường hợp ta không thể thêm mới mũi tên nào mà đỉnh đích vẫn chưa được gán nhãn thì không tồn tại đường đi từ đỉnh gốc tới đỉnh đích, nghĩa là không tồn tại đường đi ngắn nhất giữa 2 đỉnh này. Với việc gán thêm mũi tên chúng ta sẽ có được cái nhìn hết sức trực quan khi thực hiện thuật toán.

Hơn nữa chúng ta thấy rằng trong thuật toán chúng ta phải tính tổng của tất cả các nhãn khoảng cách đã có với trọng số sau mỗi bước khi chúng ta kết nạp thêm một đỉnh mới tức là tính $d(i)+c_{ij}$ với mọi đỉnh i có nhãn cố định (có mũi tên đã được lựa chon đi ra

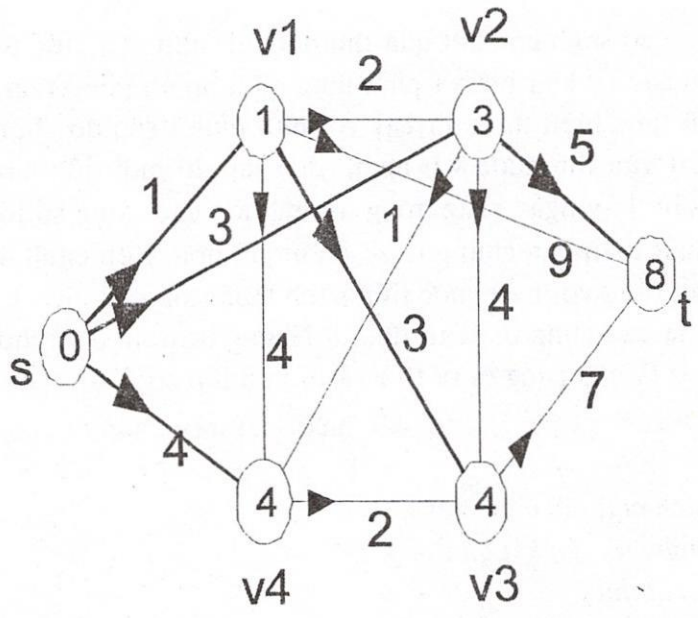
từ i) sau đó tiến hành so sánh các kết quả tìm được để tìm d(j) việc tính toán như vậy có thể dẫn đến so sánh các số khá lớn và phải thực hiện nhiều phép tính. Do đó thay vì làm như vậy chúng ta sẽ thực hiện tìm $\min\{c_{ij}\}$ với mỗi đỉnh i sau đó chúng ta mới tiến hành cộng giá trị nhỏ nhất vừa tìm được với nhãn d(i) sau đó mới tiến hành so sánh với d(j). Chúng ta có thể nhận thấy ngay rằng trong trường hợp các trọng số lớn khác nhau và đã được sắp xếp thì thuật toán của chúng ta sẽ hiệu quả hơn. Bên cạnh đó chúng ta cũng có thể có nhận xét ngay rằng với mỗi bước tiến hành thuật toán cổ điển khi chúng ta phải xét các cung $(i,j) \in A(i)$ là các cung đi ra từ đỉnh i. Nhưng trên thực tế chúng ta thấy rằng chỉ cần xét các cung (i,j) là các cung đi ra từ i và đi vào tập có nhãn thay đổi. Tức là chỉ cần xét các cung (i,j) với $i \in S$ và $j \in \overline{S}$. Với các nhận xét trên chúng ta có thuật toán được cải tiến như sau.

*CArr: danh sách mũi tên đang xét*
*RArr: danh sách mũi tên kết quả*
*d(i): Nhãn của đỉnh i*
*c(i,j): trọng số cung đi từ đỉnh i tới j*
*CostArr(i,j): trọng số của mũi tên gốc i, hướng tới đỉnh j*
*s,t: đỉnh xuất phát và kết thúc*

**Thuật toán**

begin
Stop:=false;
i:=s;
d(s):=0;
S:={s};
While (Stop=false)
begin
for $i \in S$ do
begin
Tìm min{c(i,j)} theo j được c(i,j*);
*costArr(i,j*)*:= d(i) + c(i,j*);
Thêm mũi tên (i,j*) vào danh sách *CArr*;
end;
if (*CArr*=$\phi$) then
begin
Stop=true; *RArr*:=$\phi$;
end;
Chọn trong các mũi tên (i,j*) mũi tên có trọng số min(min{*costArr*(i,j*)} theo $i \in S$) ta được (i*,j*);
Thêm mũi tên (i*,j*) vào danh sách *RArr*;
d(j*):= *costArr*(i*,j*);
S:=S$\cup${j*};
If (j=t) then stop:=true else i:=j;
Xóa các mũi tên có đỉnh đến là j khỏi danh sách *CArr*;
end;
end.

Sau đây ta sẽ xem một ví dụ để minh họa cho thuật toán .




**Hình 3**

Ví dụ chúng ta xét với giả thiết như ở ví dụ trước. Khi đó thuật toán của chúng ta sẽ được tiến hành như sau: Bước đầu tiên chúng ta lựa chọn đỉnh s và thấy ngay cung đi ra từ s có trọng số nhỏ nhất là 1 và đi tới $v_1$, sau đó chúng ta lựa chọn đỉnh $v_1$ và loại bỏ cạnh $sv_1$. Bước tiếp theo ta xét các cung có trọng số nhỏ nhất trong các cung đi ra từ đỉnh s và đỉnh $v_1$ tương ứng là các cung $sv_2$ và $v_1v_2$ với các trọng số là 3 và 2 tương ứng khi đó chúng ta xác định $\min\{d(s)+3, d(v_1)+2\}= \min\{3,3\}=3$ chúng ta chọn đỉnh $v_2$ và lựa chọn cung $sv_2$ (do đường đi ngắn nhất không duy nhất nên có thể lựa chọn cung $v_1v_2$) và loại bỏ các cạnh $sv_2$ và $v_1v_2$. Bước tiếp theo ta lại tiếp tục tìm các trọng số nhỏ nhất trong các cung đi ra từ các đỉnh $s,v_1,v_2$ tương ứng và làm tương tự chúng ta sẽ lựa chọn được đỉnh $v_4$ với cung được lựa chọ là $sv_4$ (hoặc $v_2v_4$) làm hoàn toàn tương tự chúng ta sẽ lần lượt lựa chọn được các đỉnh $v_3$, t trong các bước tiếp theo và thu được khoảng cách ngắn nhất từ s đến t là 8. Chúng ta thấy rằng trong ví dụ thực hiện chúng ta chỉ cần vẽ một hình minh hoạ và tại mỗi bước chỉ phải tính toán với các cung có trọng số nhỏ nhất thay vì việc tính với tất cả các trọng số tại mỗi bước.

Độ phức tạp của thuật toán trên thực tế trong trường hợp xấu nhất không cải tiến được nhiều so với thuật toán ban đầu nhưng trong trường hợp dữ liệu vào đã được sắp xếp và đầu vào lớn thì thuật toán của chúng ta sẽ thực sự hiệu quả hơn. Và thuật toán trên cũng có thể áp dụng cho đồ thị với trọng số thực bất kỳ không âm chứ không chỉ hạn chế trên tập trọng số tự nhiên.

## TO ENLARGE OF THE DIJKSTRA ALGORITHM

**Vu Dinh Hoa, Ta Anh Son, Đo Thi Bich Ngoc**
Deparment Informatic Technology, Ha Noi University of Pedagogy

***ABSTRACT:*** *Shortest path problems lie at the heart of network flows, They arise frequently in practice since in a wide variety of application settings we wish to send some material*

*(e.g,a computer data packet, a telephone call, a vehicle) between two specified point in a network as quickly, as cheaply, as reliably as possible...Some author is interested in this problem and the they have had many results. An algorithm was proviced by Dijkstra(1959) but now it is used in wide variety because it is simple and effective... In this paper we provice a way to use Dijkstra's algorithm to help us easy to used and have a visualization. Futhermore, in classical algorithm we must give a new label for each vertices in each step but in this new vision we use arrows for each vertices so give labels at most once times and unnecessary start label equal* $\infty$ .

## TÀI LIỆU THAM KHẢO

[1] **Claude Berge**, *Lý thuyết đồ thị và ứng dụng*, Nguyễn Hữu Nguyên và Nguyễn Văn Vy dịch, NXB Khoa học kỹ thuật, Hà nội (1971).

[2] **R.K.Ahuja, T.L.Magnanti, J.B.Orlin**, *NETWORK FLOWS* Theory, Algorithms, and Application, PRENTICE HALL,Upper Saddle River, New Jersey 07458.

[3] **Kenneth H.Rosen,** *Toán học rời rạc ứng dụng trong tin học*, Nhà xuất bản khoa học kỹ thật Hà Nội (1998).

[4] **Vũ Đình Hòa**, *Một số kiến thức cơ sở về Graph hữu hạn*. Nhà xuất bản Giáo dục, Đà nẵng, (2004).

[5] **Vũ Đình Hòa**, *Định lí và vấn đề về đồ thị hữu hạn*, Nhà xuất bản Giáo dục, Hà nội, 2002.

[6] **Vũ Đình Hòa**, *Mạng đường tối ưu*, Tuyển tập 30 năm Tạp chí Toán Học và Tuổi trẻ, trang 382-386, NXB Giáo Dục 2004.