

**BÁO CÁO MÔN HỌC**

**PHÂN TÍCH VÀ THIẾT KẾ GIẢI THUẬT**

***Đề 14:***

**Bài toán Global minimum cuts trên đồ thị với thuật toán Karger**

*Giáo viên hướng dẫn: Nguyễn Mậu Uyên*

*Sinh viên thực hiện: Nguyễn Văn Trường*

*Lớp: KHMT 14*

Hà Nội, 12/2017

**MỤC LỤC**

PHỤ LỤC

[**1.** **Bài toán** 3](#_Toc501164239)

[**2.** **Mô tả chi tiết thuật toán** 3](#_Toc501164240)

[**3.** **Ví dụ** 3](#_Toc501164241)

[**4.** **Viết chương trình trên C và kết quả** 4](#_Toc501164242)

[**5. Đánh giá độ phức tạp thuật toán** 9](#_Toc501164243)

# **Bài toán**

Xét một đơn đồ thị có hướng hoặc vô hướng G= (V, E).

Ý tưởng của bài toán là hợp nhất 2 đỉnh của của một cạnh e trong đồ thị G= (V, E) cho đến khi đồ thị trở thành 2 phần liên thông.

# **Mô tả chi tiết thuật toán**

1. Ý tưởng

Trong thuật toán này:

-Đầu tiên ta chọn bất kì một cạnh nào đó ở đây ta dùng thuật toán random.

-Rồi khi có cạnh đó rồi thì ta sẽ phải tìm 2 đỉnh của cạnh đó và tìm nguồn gốc hoặc là cha mẹ của đỉnh đó.

-Mỗi đỉnh sẽ có một biến để lưu lại gốc (hoặc cha mẹ) của đỉnh đó, mỗi đỉnh sẽ chỉ có một parent.

-Nếu 2 đỉnh này đã hợp nhất với nhau rồi thì nó sẽ có cùng gốc (cha mẹ), nếu cùng cha mẹ thì sẽ thoát ra và random để có cạnh khác, sau khi hợp nhất số cạnh sẽ trừ đi 1.

-Sau đó ta sẽ đi từ cạnh thứ 1 đến cạnh thứ n để tìm ra số cạnh bị cắt đi mà nhỏ nhất có thể, nếu cạnh đó có 2 đỉnh chung một gốc thì cạnh đó đã bị cắt.

1. Mô tả chi tiết thuật toán

-đầu tiên ta viết một struct để để định nghĩa một cạnh

-một cạnh sẽ gồm 2 đỉnh

// một struct để lưu cạnh gồm 2 đỉnh!

struct Edge

{

int DinhA,DinhB;

};

-Tiếp theo đó ta định nghĩa một struct để định nghĩa một Đồ thị

-Đồ thị này gồm có số đỉnh V và số cạnh E

-Và bên trong có các cạnh của đồ thị

//một struct để thể hiện đồ thị gồm V đỉnh và E cạnh

struct Graph

{

//V-> số đỉnh E ->là số cạnh

int V, E;

Edge\* edge;

};

-Tiếp theo là một struct để thuận tiện cho việc tìm kiếm gốc hoặc cha của đỉnh đó là một biến có tên là parent và cũng với đó là một biến rank.

-Nếu một đỉnh có rank cao thì tức là đỉnh đó có nhiều đỉnh con hợp vào

// một struct để lưu trữ khi việc tìm và nối đỉnh

struct Subset {

int parent;

int rank;

};

-Tiếp theo là ta viết một hàm tìm gốc của đỉnh tức là tìm cha của đỉnh đó.

-Lúc đầu ta sẽ dùng một vòng for để gán chính nó là gốc của nó và có rank bằng 0.

int TimGocCanh(struct Subset subsets[], int i)

{

//tìm gốc của đỉnh đang xét

//nếu gốc của nó không còn nó như lúc đầu thiết lập nữa thì ta sẽ dùng đệ quy gọi lại hàm này để tìm gốc thật sự của đỉnh đó

if (subsets[i].parent != i)

{

subsets[i].parent = TimGocCanh(subsets, subsets[i].parent);

}

return subsets[i].parent;

}

-Ta viết một hàm để nối 2 đỉnh X và Y với nhau

//hàm để nối hai đỉnh x và y với nhau(ưu tiên nối vào đỉnh có rank cao hơn)

void NoiDinh(struct Subset subsets[], int x, int y)

{

int xroot = TimGocCanh(subsets, x);

int yroot = TimGocCanh(subsets, y);

//nếu rank của đỉnh y cao hơn thì hợp vào y

if (subsets[xroot].rank < subsets[yroot].rank)

subsets[xroot].parent = yroot;

//ngược lại nếu rank của x cao hơn thì hợp vào x

else if(subsets[xroot].rank > subsets[yroot].rank)

subsets[yroot].parent = xroot;

//nếu bằng nhau thì ta sẽ cho hợp với x và tăng rank x lên 1 đơn vị

else {

subsets[yroot].parent = xroot;

subsets[xroot].rank++;

}

}

-Đây là thân hàm của thuật toán Karger

int kargerMinCut(struct Graph\* graph)

{

// tạo dữ liệu để chứa Graph

int V = graph->V, E = graph->E;

Edge \*edge = graph->edge;

// Tạo mảng để lưu trữ V Subsets

struct Subset \*subsets = new Subset[V];

//khởi tạo giá trị cho con phần tử trong mảng Subsets

for (int v = 0; v < V; ++v)

{

subsets[v].parent = v;

subsets[v].rank = 0;

}

int SoDinh = V;

//tạo một vòng lặp while chạy đến bao giờ chỉ còn lại 2 đỉnh

while (SoDinh>2)

{

//chọn một cạnh bất kì

int i = rand() % E;

//tìm đỉnh của 2 đầu cạnh

int subset1 = TimGocDinh(subsets, edge[i].DinhA);

int subset2 = TimGocDinh(subsets, edge[i].DinhB);

//nếu 2 đầu cạnh bằng nhau thì không xét cạnh này

if (subset1 == subset2)

continue;

//ngược lại ta sẽ hợp 2 đỉnh của cạnh này với nhau

else

{

printf("Noi dinh %d-%d\n",

edge[i].DinhA,edge[i].DinhB);

SoDinh--;

NoiDinh(subsets, subset1, subset2);

}

}

int cutedges = 0;

//để biết được có bao cạnh bị cắt thì ta sẽ gọi lại mỗi cạnh nếu 2 đầu của cạnh có chung một gốc thì nó sẽ tăng biến đếm lên 1 đơn vị

for (int i = 0; i<E; i++)

{

int subset1 = TimGocDinh(subsets, edge[i].DinhA);

int subset2 = TimGocDinh(subsets, edge[i].DinhB);

if (subset1 == subset2)

cutedges++;

}

return cutedges;

}

-Ta cũng tạo một hàm để khởi tạo một graph với V và E

struct Graph\* TaoGraph(int V,int E)

{

Graph\* graph = new Graph;

graph->V = V;

graph->E = E;

graph->edge = new Edge[E];

return graph;

};

# **Ví dụ**

Chuẩn bị 2 bộ dữ liệu

* V={0,1,2,3,4,5}
* E=8 cạnh

Bộ dữ liệu thứ 2

-V={0,1,2,3,4,5}

-E=9

# **Viết chương trình trên C và kết quả**

#include<stdio.h>

#include<conio.h>

#include<time.h>

#include<stdlib.h>

// một struct để lưu cạnh gồm 2 đỉnh!

struct Edge

{

int DinhA,DinhB;

};

//một struct để thể hiện đồ thị gồm V đỉnh và E cạnh

struct Graph

{

//V-> số đỉnh E ->là số cạnh

int V, E;

Edge\* edge;

};

// một struct để lưu trữ khi việc tìm và nối đỉnh

struct Subset {

int parent;

int rank;

};

int TimGocDinh(struct Subset subsets[], int i);

void NoiDinh(struct Subset subsets[],int x,int y);

int kargerMinCut(struct Graph\* graph)

{

// tạo dữ liệu để chứa Graph

int V = graph->V, E = graph->E;

Edge \*edge = graph->edge;

// Tạo mảng để lưu trữ V Subsets

struct Subset \*subsets = new Subset[V];

//khởi tạo giá trị cho con phần tử trong mảng Subsets

for (int v = 0; v < V; ++v)

{

subsets[v].parent = v;

subsets[v].rank = 0;

}

int SoDinh = V;

//tạo một vòng lặp while chạy đến bao giờ chỉ còn lại 2 đỉnh

while (SoDinh>2)

{

//chọn một cạnh bất kì

int i = rand() % E;

//tìm đỉnh của 2 đầu cạnh

int subset1 = TimGocDinh(subsets, edge[i].DinhA);

int subset2 = TimGocDinh(subsets, edge[i].DinhB);

//nếu 2 đầu cạnh bằng nhau thì không xét cạnh này

if (subset1 == subset2)

{

printf("Canh Random co 2 dinh la :%d - %d(Da hop nhat)\n", edge[i].DinhA, edge[i].DinhB);

continue;

}

//ngược lại ta sẽ hợp 2 đỉnh của cạnh này với nhau

else

{

printf("Hop nhat hai dinh : %d-%d\n",

edge[i].DinhA,edge[i].DinhB);

SoDinh--;

NoiDinh(subsets, subset1, subset2);

}

}

int cutedges = 0;

for (int i = 0; i<E; i++)

{

int subset1 = TimGocDinh(subsets, edge[i].DinhA);

int subset2 = TimGocDinh(subsets, edge[i].DinhB);

if (subset1 == subset2)

cutedges++;

}

return cutedges;

}

int TimGocDinh(struct Subset subsets[], int i)

{

//tìm gốc của đỉnh đang xét

//nếu gốc

if (subsets[i].parent != i)

{

subsets[i].parent = TimGocDinh(subsets, subsets[i].parent);

}

return subsets[i].parent;

}

//hàm để nối hai đỉnh x và y với nhau(ưu tiên nối vào đỉnh có rank cao hơn)

void NoiDinh(struct Subset subsets[], int x, int y)

{

int xroot = TimGocDinh(subsets, x);

int yroot = TimGocDinh(subsets, y);

//nếu rank của đỉnh y cao hơn thì hợp vào y

if (subsets[xroot].rank < subsets[yroot].rank)

subsets[xroot].parent = yroot;

//ngược lại nếu rank của x cao hơn thì hợp vào x

else if(subsets[xroot].rank > subsets[yroot].rank)

subsets[yroot].parent = xroot;

//nếu bằng nhau thì ta sẽ cho hợp với x và tăng rank x lên 1 đơn vị

else {

subsets[yroot].parent = xroot;

subsets[xroot].rank++;

}

}

//hàm khởi tạo một đồ thị có V đỉnh và E cạnh

struct Graph\* TaoGraph(int V,int E)

{

Graph\* graph = new Graph;

graph->V = V;

graph->E = E;

graph->edge = new Edge[E];

return graph;

};

int main()

{

int V = 6;

int E = 9;

struct Graph\* graph = TaoGraph(V, E);

// thêm edge 0-1

graph->edge[0].DinhA= 0;

graph->edge[0].DinhB = 1;

// thêm edge 0-2

graph->edge[1].DinhA = 0;

graph->edge[1].DinhB = 2;

// thêm edge 0-3

graph->edge[2].DinhA = 0;

graph->edge[2].DinhB = 3;

// thêm edge 1-3

graph->edge[3].DinhA = 1;

graph->edge[3].DinhB = 3;

// thêm edge 2-3

graph->edge[4].DinhA = 2;

graph->edge[4].DinhB = 3;

//thêm edge 1-4

graph->edge[5].DinhA = 1;

graph->edge[5].DinhB = 4;

//thêm edge 4-5

graph->edge[6].DinhA = 4;

graph->edge[6].DinhB = 5;

//thêm edge 3-5

graph->edge[7].DinhA = 3;

graph->edge[7].DinhB = 5;

//thêm edge 4-5

graph->edge[8].DinhA = 4;

graph->edge[8].DinhB = 3;

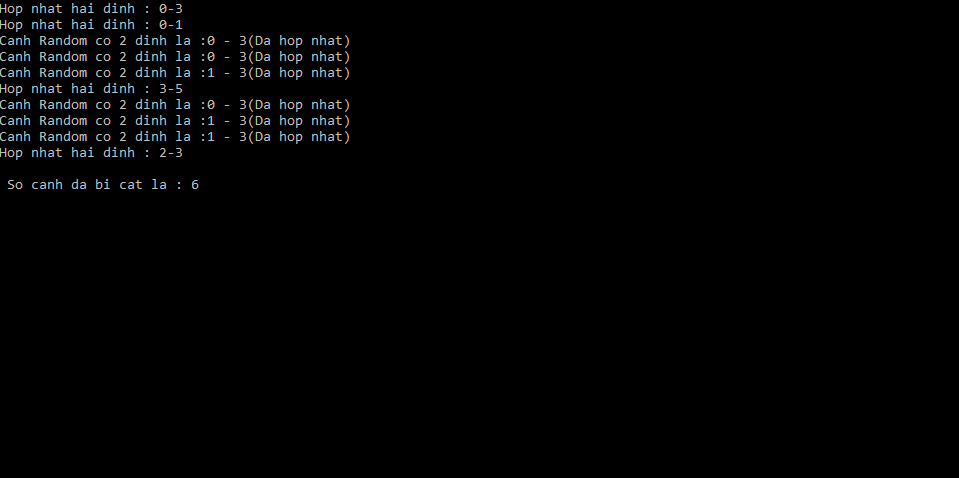
srand(time(NULL));

printf("\n So canh da bi cat la : %d\n",

kargerMinCut(graph));

\_getch();

return 1;

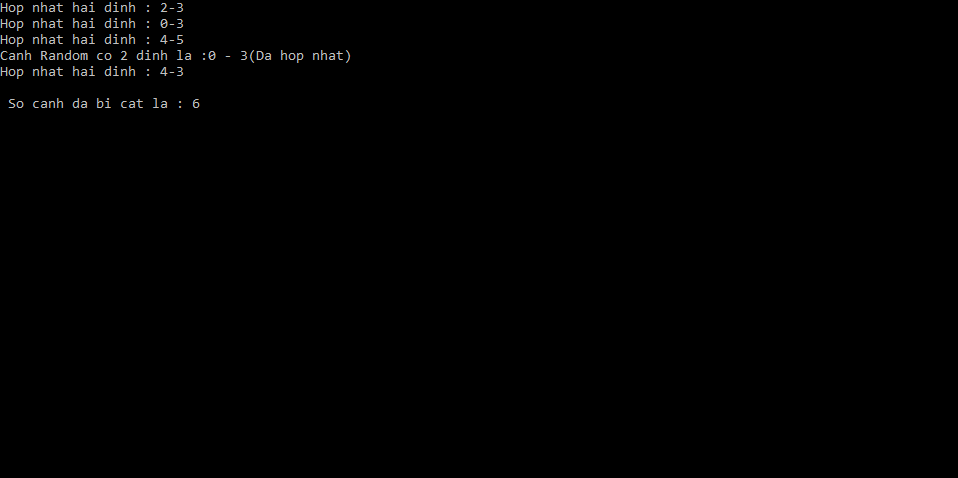
}

-Kết quả của bộ dữ liệu thứ nhất :

Kết quả trả về cho ta thấy

* Sau random ta có cạnh 2 đỉnh là 0-3 vì lúc này chưa có đỉnh nào hợp nhất với nhau cả cho nên ta hợp nhất 2 đỉnh này với nhau.Lần tiếp theo random là 0-1 là tiếp tục hợp nhất.
* Đến lần thứ 3 thì chúng ta random ra 2 đỉnh 0-3 đã hợp nhất vậy nên ta bỏ qua và tiếp tục random.Tiếp theo là ra 1-3 vì 0 hợp với 3 mà 1 lại hợp với 0 vậy nên 1 đã hợp với 3.
* -Cứ như vậy ta có tiếp theo cạnh có định 3-5 và cạnh đỉnh 2-3 kết hợp 2 đỉnh với nhau.
* Kết quả đưa ra là có 6 cạnh bị cắt để hợp nhất các đỉnh.

Kết quả của bộ dữ liệu thứ 2:



1. Cũng như trên khi sử dụng bộ dữ liệu thứ 2 này thì ta thu được kết quả
2. Hợp nhất đinh 2-3,0-3,4-5,4-3
3. Trả về số cạnh bị cắt là 6.

# **5.** **Đánh giá độ phức tạp thuật toán**

Để tính được độ phức tạp của thuật toán thì ta cần lắm rõ các quy tắc sau:

Độ phức tạp của giải thuật: O(f(n))

Việc xác định độ phức tạp của giải thuật thực tế có thể tính bằng một số quy tắc đơn gản sau:

- Quy tắc bỏ hằng :

T(n) = O(c.f(n)) = O(f(n)) với c là một hằng số dương

- Quy tắc lấy max :

T(n)=O(c.f(n)+g(n))=O(max(f(n),g(n)))

- Quy tắc cộng :

T1(n) = O(f(n))

T2(n) = O(g(n))

T1(n) + T2(n) = O(f(n) + g(n))

- Quy tắc nhân :

Đọan chương trình có thời gian thực hiện T(n)=O(f(n))

Nếu thực hiện k(n) lần đoạn chương trình với k(n)=O(g(n)) thì độ phức tạp là O(g(n).f(n))

**Cài đặt chương trình như ở mục 4. Ta có:**

int kargerMinCut(struct Graph\* graph)

{

// tạo dữ liệu để chứa Graph

int V = graph->V, E = graph->E; ->2 phép gán

Edge \*edge = graph->edge; ->1 phép gán

// Tạo mảng để lưu trữ V Subsets

struct Subset \*subsets = new Subset[V]; ->phép gán

//khởi tạo giá trị cho con phần tử trong mảng Subsets

for (int v = 0; v < V; ++v)

{

subsets[v].parent = v; ->4+5\*(V-1) phép gán

subsets[v].rank = 0;

}

int SoDinh = V; ->phép gán

//tạo một vòng lặp while chạy đến bao giờ chỉ còn lại 2 đỉnh

while (SoDinh>2) ->so sánh

{

//chọn một cạnh bất kì

int i = rand() % E; ->gán

//tìm đỉnh của 2 đầu cạnh

int subset1 = TimGocDinh(subsets, edge[i].DinhA);

int TimGocDinh(struct Subset subsets[], int i)

{

//tìm gốc của đỉnh đang xét

//nếu gốc

if (subsets[i].parent != i)

{

subsets[i].parent = TimGocDinh(subsets, subsets[i].parent);

}

return subsets[i].parent;

}

->1 phép trả về

->2 phép gán

Vậy Có tối đa V-1 lần gọi đệ quy ->> tối đa 3+2\*(V-2) + 1

int subset2 = TimGocDinh(subsets, edge[i].DinhB);

->có tối đa 3+2\*(V-2) +1

//nếu 2 đầu cạnh bằng nhau thì không xét cạnh này

if (subset1 == subset2) ->1 phép so sánh

continue; ->1 continue

//ngược lại ta sẽ hợp 2 đỉnh của cạnh này với nhau

else

{

printf("Noi dinh %d-%d\n",

edge[i].DinhA,edge[i].DinhB);

SoDinh--; ->1 phép gán

NoiDinh(subsets, subset1, subset2);

void NoiDinh(struct Subset subsets[], int x, int y)

{

int xroot = TimGocDinh(subsets, x);

int yroot = TimGocDinh(subsets, y);

//nếu rank của đỉnh y cao hơn thì hợp vào y

if (subsets[xroot].rank < subsets[yroot].rank)

subsets[xroot].parent = yroot;

//ngược lại nếu rank của x cao hơn thì hợp vào x

else if(subsets[xroot].rank > subsets[yroot].rank)

subsets[yroot].parent = xroot;

//nếu bằng nhau thì ta sẽ cho hợp với x và tăng rank x lên 1 đơn vị

else {

subsets[yroot].parent = xroot;

subsets[xroot].rank++;

}

->1+3+2\*(V-2)

-> mỗi trường hợp có thêm 2 phép gán nữa

}

}

}

int cutedges = 0; ->gán

for (int i = 0; i<E; i++)

{

int subset1 = TimGocDinh(subsets, edge[i].DinhA);

->1+3+2\*(V-2)

int subset2 = TimGocDinh(subsets, edge[i].DinhB);

->1+3+2\*(V-2)

if (subset1 == subset2) ->1 so sánh

cutedges++; ->1 gán

}

-Độ phức tạp trong dòng for này thì E\*(2\*(1+3+2\*(V-2)) + 2)

return cutedges; ->1 trả về

}

Đề tổng kết lại độ phức tạp của thuật toán này là:

=

=

* **Vậy độ phức tạp của thuật toán là**