HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG CƠ SỞ TẠI
THÀNH PHỐ HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN 2

BÁO CÁO
ĐỒ ÁN MÔN HỌC

ĐỀ TÀI: GIẢI BÀI TOÁN GHÉP CẶP TRÊN ĐỒ THỊ (MATCHING)

Môn học: Cấu trúc dữ liệu và giải thuật

Giảng viên: Lê Văn Hạnh

Sinh viên thực hiện:

Huỳnh Thị Hồng Thắm N23DCCN055 n23dccn055@student.ptithcm.edu.vn

TP.HCM, tháng 05/2025

MỤC LỤC

DANH SÁCH HÌNH ẢNH

TÓM TẮT

Bài báo cáo này trình bày về thuật toán Edmonds' Blossom tìm cặp ghép cực đại trên đồ thị tổng quát. Thuật toán cài đặt bằng ngôn ngữ C++ chỉ sử dụng thư viện chuẩn <iostream>. Bên cạnh lí thuyết còn có mô tả chi tiết từng thuật toán nhỏ thuộc phạm trù Edmonds' Blossom dùng trong chương trình tìm các cặp ghép cực đại.

I/ TỔNG QUAN VỀ ĐỀ TÀI

1.1 Giới thiệu về Ghép cặp trên đồ thị (Matching)

Bài toán ghép cặp trên đồ thị là một trong những bài toán quan trọng trong lý thuyết đồ thị, với nhiều ứng dụng thực tế trong phân công công việc, ghép cặp sinh viên với đề tài nghiên cứu, cũng như tìm cặp phù hợp trong hệ thống gợi ý.

Ghép cặp (Matching) trong đồ thị là một tập hợp các cạnh sao cho không có hai cạnh nào có chung một đỉnh.

Thuật toán sử dụng trong đề tài : Edmonds' Blossom

Thuật toán ghép cặp của Edmonds (còn được gọi là thuật toán bông hoa – Blossom Algorithm) là một thuật toán dùng để tìm matching cực đại trong đồ thị tổng quát. Thuật toán này được đề xuất bởi Jack Edmonds 1961 và được xuất bản chính thức năm 1965.

Cho một đồ thị vô hướng G=(V,E), thuật toán tìm một tập ghép M sao cho mỗi đỉnh trong V kề với nhiều nhất một cạnh trong M, và kích thước |M| là lớn nhất có thể. Matching khởi đầu từ tập rỗng, sau đó được mở rộng thông qua các đường tăng (augmenting paths).

Điểm đột phá trong thuật toán này là việc xử lý các chu trình lẻ (odd cycles). Trong quá trình tìm đường tăng, thuật toán sẽ thu gọn (contract) toàn bộ chu trình đó lại thành một đỉnh duy nhất (blossom), từ đó tiếp tục tìm kiếm trên đồ thị đã được thu gọn.

Thuật toán Edmonds là chứng minh đầu tiên cho thấy bài toán tìm matching cực đại có thể giải được trong thời gian đa thức (polynomial time). Một đóng góp quan trọng khác của thuật toán là giúp xây dựng và mô tả được đa diện matching (matching polytope) — nền tảng cho việc phát triển các thuật toán tìm matching trọng số nhỏ nhất.

Ứng dụng thực tế

1. Phân công công việc: Blossom giúp tìm cách ghép các nhân viên làm các công việc, mỗi thành viên tập trung vào phần phù hợp với năng lực, giúp đẩy nhanh tiến độ.
2. Ghép cặp sinh viên – đề tài: Khi sinh viên đăng ký các đề tài nghiên cứu, mỗi đề tài có một số sinh viên quan tâm. Blossom giúp phân chia mỗi sinh viên một cách tối ưu để làm đề tài.

3. Hệ thống gợi ý: Trong các ứng dụng hẹn hò, tuyển dụng hay thương mại điện tử, hệ thống gợi ý giúp tìm ra sự ghép cặp tốt nhất giữa các đối tượng dựa trên sở thích hoặc đặc điểm chung.

1.2 Cơ sở lý thuyết

1.2.1 Đồ thị (Graph)

Đồ thị vô hướng (Undirected Graph): Trong đồ thị vô hướng, mối quan hệ giữa hai đỉnh bất kì là hai chiều. Nếu có một cạnh nối đỉnh A và đỉnh B, thì có thể di chuyển từ A đến B và ngược lại. Ví dụ, một mạng lưới giao thông giữa các thành phố mà không có đường một chiều có thể được biểu diễn bằng đồ thị không hướng.

Đồ thị có hướng (Directed Graph): Trong đồ thị có hướng, mối quan hệ giữa hai đỉnh là một chiều. Nếu có một cạnh hướng từ đỉnh A đến đỉnh B, có thể di chuyển từ A đến B, nhưng không thể di chuyển ngược lại (trừ khi có một cạnh hướng từ B đến A). Ví dụ, một sơ đồ dòng chảy công việc, nơi mỗi bước phải hoàn thành trước khi chuyển sang bước tiếp theo, có thể được biểu diễn bằng đồ thị có hướng.

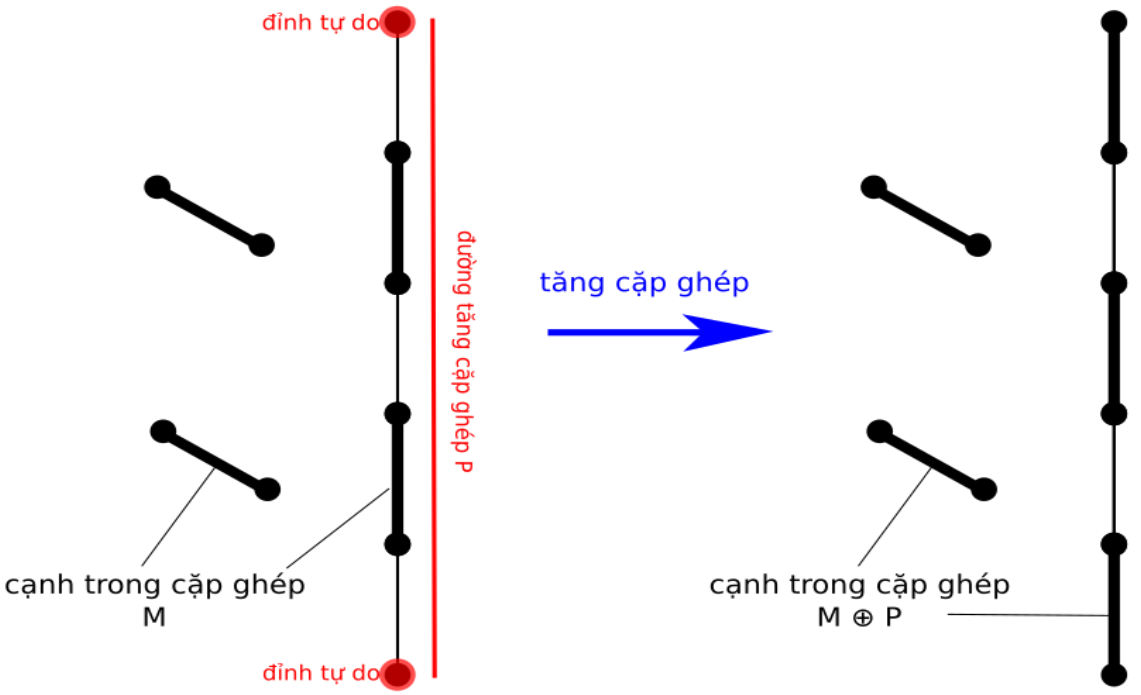
1.2.2 Đường tăng (Augmenting path)

Là đường đi từ đỉnh nguồn đến đỉnh đích trên đồ thị có thể chứa các cạnh được tăng giá trị luồng(bằng lượng tối đa có thể được tăng).

Ta có đồ thị G=(V, E) và cặp ghép M của đỉnh G. Nếu không có cạnh nào của M kề với v thì v tự do, nếu cách cạnh của G luân phiên nhau nằm trong M và không nằm trong M thì một đường trong G là đường luân phiên. Một đường tăng là đường luân phiên bắt đầu và kết thúc tại hai đỉnh tự do. Phép tăng cặp ghép dọc theo đường tăng là việc thay M bằng cặp ghép

$$M_1=M \oplus P=(M \setminus P) \cup (P \setminus M)$$

Đây là phép XOR giữa M và P trên tập cạnh (được gọi là phép đối xứng hiệu). Mỗi cặp cạnh trong P sẽ được "lật trạng thái": nếu đang thuộc M thì loại bỏ, nếu không thuộc thì thêm vào.



Hình 1: Minh họa tăng cặp ghép

Cặp ghép M là cực đại khi và chỉ khi không tồn tại đường tăng cho M trong. Do đó, hoặc một cặp ghép là cực đại, hoặc tồn tại đường tăng cho nó.

```
Dữ liệu vào:  Đồ thị G, cặp ghép ban đầu M trên G
Dữ liệu ra: cặp ghép cực đại M* trên G
function tìm_cặp_ghép_cực_đại(G, M): M*
    P ← tìm_đường_tăng(G, M)
    if P khác rỗng
        return tìm_cặp_ghép_cực_đại(G, tăng M dọc theo P)
    else
        return M
```

Hình 2: Thuật toán tìm cặp ghép cực đại

Dòng 2: Tìm một đường tăng P trong đồ thị G với matching hiện tại M

Dòng 3: Nếu tồn tại đường tăng P, thì cải thiện M bằng cách tăng dọc theo P

Dòng 4: Đệ quy: Nếu cạnh chưa thuộc M đưa vào M* và đã thuộc thì loại ra khỏi M

Dòng 5-6: Nếu không còn đường tăng nào, trả về M, vì theo định lý Berge, M lúc này là một cặp ghép cực đại

```
Dữ liệu vào: Đồ thị G, cặp ghép M trên G
Dữ liệu ra: đường tăng P trên G hoặc thông báo không tồn tại đường tăng
function tìm_đường_tăng(G, M): P
   F ← rừng rỗng
   gán nhãn chưa thăm mọi đỉnh và mọi cạnh của G, gán nhãn đã thăm cho mọi cạnh trong M
   for each đỉnh tự do v
       tạo một cây chỉ gồm đúng một đỉnh { v } và thêm nó vào F
   while tồn tại đỉnh v chưa được thăm trong F với khoảng_cách(v, gốc(v)) là chẵn
       while tồn tại cạnh chưa thăm e = { v, w }
           if w không nằm trong F
               // Cập nhật F.
               x ← đỉnh ghép với w trong M
               thêm cạnh { v, w } và { w, x } vào cây của v
           else
           if khoảng_cách(w, gốc(w)) là lẻ
               không làm gì
           else
           if gốc(v) ≠ gốc(w)
                // trả về đường tăng trong F ∪ { e }.
               P ← đường (gốc(v) →... → v) → (w →... → gốc(w))
               return P
           else
               // thu gọn bông hoa trong G và tìm đường tăng trong đồ thị thu gọn.
               B ← bông hoa tạo bởi e và các cạnh trên đường v → w trong T
               G', M' ← thu gọn G và M bằng B
               P' ← tìm_đường_tăng(G', M')
               P ← nâng P' lên đường tăng trong G
               return P
           end if
           đánh dấu đã thăm e
       end while
       đánh dấu đã thăm v
   end while
   return không tồn tại đường tăng
end function
```

Hình 3: Thuật toán tìm đường tăng

Dòng 2-5: Tạo rừng F với các đỉnh chưa ghép. Mỗi đỉnh là gốc mỗi cây

Dòng 6-7: Duyệt các đỉnh có khoảng cách chẵn đến các đỉnh chưa ghép, mở rộng cây qua các cạnh chưa thăm

Dòng 8-11: w chưa nằm trong rừng => nối w và x vào cây (w thuộc M)

Dòng 13-14: w trong rừng và khoảng cách lẻ tới gốc=> không tạo đường tăng

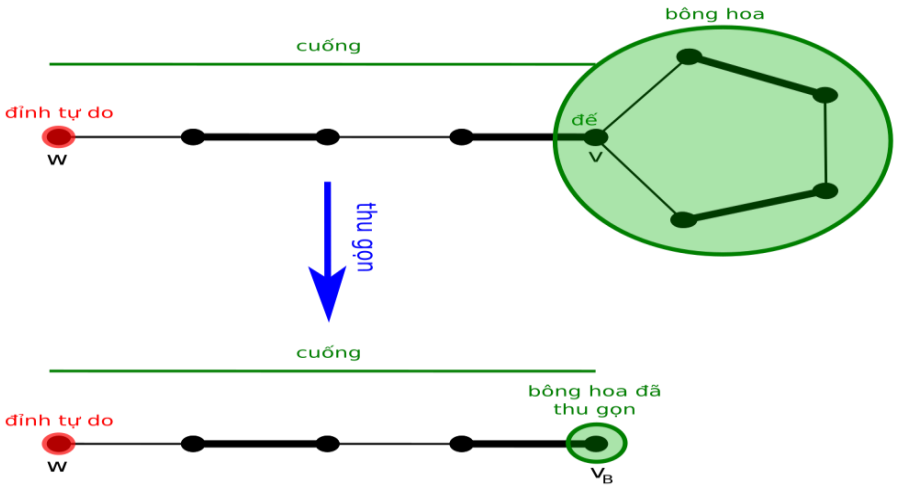
Dòng 16-19: Nếu gốc(v) ≠ gốc(w) và ở mức chẵn => tìm được đường tăng

Dòng 21-26: Nếu gốc(v) = gốc(w) ⇒ có chu trình lẻ(blosson) => thu gọn, đệ quy tìm đường tăng trong đó, rồi nâng lên lại

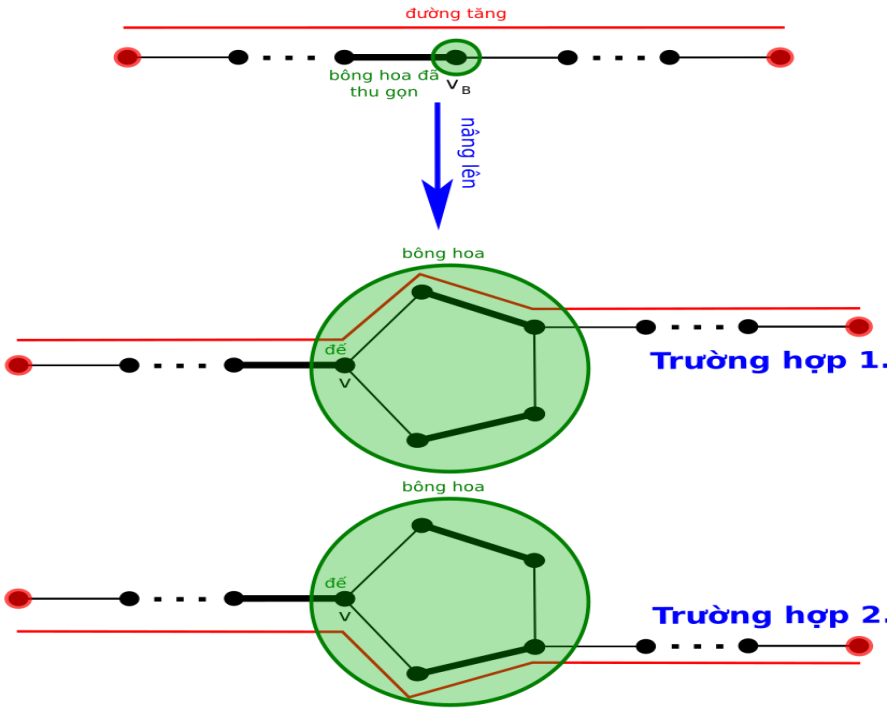
Dòng 32: Không tìm được đường tăng => trả ra "không tồn tại"

1.2.3 Blossoms and contractions

Xét đồ thị G = (V, E) và cặp ghép M. Một bông hoa A gồm 2k+1 cạnh có k cạnh nằm trong M, một đỉnh v(đế) sao cho có một đường luân phiên đồ dài chẵn(cuống) từ v đến một đỉnh w. Thu gọn toàn bộ B vào đỉnh đế hoa được đồ thị thu gọn G'.



Hình 4: Minh họa thu gọn



Hình 5: Minh họa nâng lên

Nếu p' đi qua hai cạnh u →v_b →w thì mở rộng thành u → (u' →… →w') → w trong G, u' w' là hai đỉnh và trogn hai nửa của B đi từ u' đến w', chọn

nửa duy trì tính luân phiên của đường đi. Từ đó có thể chứng minh G' có đường tăng cho M' khi và chỉ khi G có đường tăng cho M và mỗi đường tăng P' cho M' trong G' có thể được nâng lên thành một đường tăng cho M trong G.

1.2.4 Cặp ghép cực đại (Maximum Matching)

Cặp ghép là tập các cạnh không có hai cạnh nào có chung đỉnh. Cặp ghép cực đại là cặp ghép có số lượng cạnh lớn nhất có thể. Trong đồ thị tổng quát, việc tìm cặp ghép cực đại phức tạp hơn do tồn tại các chu trình lẻ (Blossom).

Thuật toán tổng quát:

```
function edmonds_matching(G):
   match[v] ← -1 với mọi v ∈ V
   for v ∈ V:
      if match[v] == -1:
         if bfs(v):
            tăng số lượng cặp ghép
   return match
```

1.2.5 Lý thuyết về thuật toán BFS

Hàm bfs có vai trò tìm đường tăng bắt đầu từ một đỉnh tự do trong đồ thị có chu trình lẻ.

Bắt đầu từ một đỉnh tự do chưa được ghép. Tìm đường luân phiên từ đỉnh tự do này đến một đỉnh tự do khác. Nếu gặp chu trình lẻ (blossom), dùng blossom_contraction() để thu gọn và tiếp tục BFS. Nếu tìm được đường tăng, cập nhật cặp ghép (match[]) dọc theo đường đó.

Cấu trúc cây trong BFS:

Các đỉnh trong cây tìm kiếm được phân theo lớp chẵn/lẻ dựa trên khoảng cách đến gốc:

Đỉnh chẵn: đi qua số chẵn cạnh → có thể mở rộng.

Đỉnh lẻ: đi qua số lẻ cạnh → đã có ghép.

```
 while queue không rỗng:
    u = pop_queue();
    for v là hàng xóm của u:
```

if v không hợp lệ → continue;

if gặp chu trình:

gọi blossom_contraction(u, v);

else nếu v chưa có cha:

nếu v chưa ghép:

tìm được đường tăng → cập nhật match[] → return true;

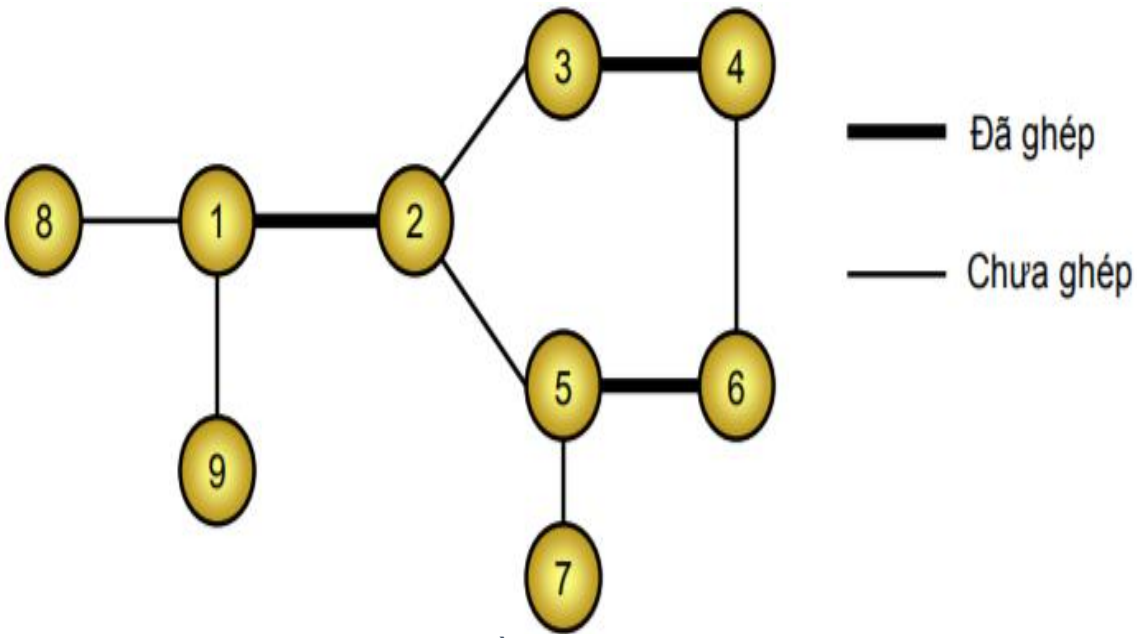
else:

parent[match[v]] = v;

push_queue(match[v]);

return false nếu không tìm thấy đường tăng;

1.2.6 Ví dụ minh họa



Hình 6: Đồ thị G và bộ ghép M

(8, 1, 2, 5, 6, 4) là đường pha (là đường bắt đầu từ cạnh nhạt(không thuộc M) tới cạnh đậm(thuộc M), nhạt và đậm nằm xen kẽ nhau).

(2, 3, 4, 6, 5, 2) là Blossom (đi qua ít nhất 3 đỉnh, bắt đầu kết thúc cạnh nhạt và dọc trên chu trình, các đường đậm nhạt nằm xen kẽ và nối tiếp nhau).

(8, 1, 2, 3, 4, 6, 5, 7) là đường mở (là đường pha bắt đầu và kết thúc ở đỉnh chưa ghép).

II/ THIẾT KẾ THUẬT TOÁN

2.1 Thuật toán Edmonds (Blossom Algorithm)

B1: Khởi tạo các đỉnh chưa được ghép

B2: Dùng BFS để tìm đường tăng(nếu tìm thấy, cập nhật cạnh ghép)

B3: Không còn đường tăng →trả về cặp ghép cực đại

Các ý tưởng chính:

Tìm đường tăng (augmenting path) là đường đi bắt đầu và kết thúc tại các đỉnh chưa ghép, luân phiên giữa các cạnh không ghép và ghép. Nếu tìm được, ta nâng cấp cặp ghép hiện tại bằng cách đổi trạng thái ghép của các cạnh trên đường đó.

BFS để tìm đường tăng: Duyệt đồ thị từ một đỉnh tự do bằng thuật toán BFS có luân phiên.

Xử lý chu trình lẻ (Blossom): Nếu trong quá trình BFS phát hiện chu trình lẻ, sẽ rơi vào vòng lặp vô hạn.Ta thu gọn bông hoa (odd-length blossom) thành một siêu đỉnh, tiếp tục BFS trên đồ thị đã thu gọn.

Sau khi tìm được đường tăng trên đồ thị thu gọn, ta nâng đường tăng ngược trở lại đồ thị ban đầu bằng cách giải nén bông hoa.

2.2 Thiết kế

2.2.1 Mục tiêu

Thuật toán Edmonds' Blossom tìm matching cực đại trong đồ thị tổng quát, có thể có chu trình lẻ. Nó mở rộng ý tưởng đường tăng trong thuật toán tìm cặp ghép.Khi gặp chu trình lẻ (blossom), nó co chu trình lại như một đỉnh và tiếp tục BFS.

2.2.2 Các hàm sử dụng

a) dayvaohangdoi(int x): Thêm đỉnh x vào hàng đợi để xử lý trong BFS. Ghi nhận rằng đỉnh x đã có trong hàng đợi (tronghangdoi [x] = true) và thêm x vào mảng hangdoi ở cuối hàng đợi.
b) layrahangdoi(): Lấy một đỉnh từ đầu hàng đợi. Trả về phần tử ở đầu mảng hangdoi, sau đó tăng chỉ số dau.
c) timlca(int a, int b): Tìm tổ tiên chung thấp nhất (LCA) giữa hai đỉnh a và b trong quá trình xây dựng đường tăng. Lần lượt đi ngược theo các cặp ghép và cha của a để đánh dấu các gốc (goc) đã đi qua. Sau đó đi ngược từ b, đỉnh đầu tiên mà trùng với đường của a chính là LCA.

d) danhdaublossom(int u, int v, int lca): Đánh dấu các đỉnh thuộc blossom (chu trình lẻ cần rút gọn) trên đường từ u về lca. Đánh dấu các đỉnh trên đường đi từ u về lca là thuộc blossom. Cập nhật cha của các đỉnh này để tạo đường tăng.
e) rutgonblossom(int u, int v): Rút gọn chu trình blossom khi phát hiện có chu trình lẻ trong quá trình tìm đường tăng. Tìm lca giữa u và v. Đánh dấu các đỉnh thuộc blossom từ cả hai phía. Cập nhật gốc (goc[i] = lca) cho tất cả các đỉnh thuộc blossom. Nếu đỉnh đó chưa có trong hàng đợi, đẩy vào để tiếp tục BFS.
f) bfs(int gocr): Tìm một đường tăng bắt đầu từ đỉnh gocr chưa được ghép cặp.Khởi tạo mảng cha, goc, tronghangdoi. Nếu gặp blossom, gọi rutgonblossom(). Nếu gặp đỉnh chưa ghép (ghep[v] == -1), thì tìm đường tăng và cập nhật ghép ngược lại từ v → gốc. Nếu gặp đỉnh đã ghép, tiếp tục mở rộng tìm kiếm bằng cách đưa đỉnh ghép vào hàng đợi.
g) edmonds(): Tìm số lượng cặp ghép cực đại trong đồ thị. Gán tất cả các đỉnh là chưa ghép (ghep[i] = -1). Duyệt từng đỉnh chưa ghép, nếu tìm được đường tăng từ BFS thì tăng số lượng ghép (dem++).
h) main(): Nhập dữ liệu đồ thị và gọi thuật toán. Nhập số đỉnh sodinh, số cạnh socanh, sau đó nhập các cạnh. Gọi edmonds() để tính số cặp ghép cực đại. In ra số cặp ghép, và danh sách các cặp.

III/ TRIỂN KHAI THUẬT TOÁN

3.1 Viết chương trình C++

```
#include <iostream>
using namespace std;
const int MAXN = 1000;
int ke[MAXN][MAXN];
int ghep[MAXN], goc[MAXN], cha[MAXN];
bool tronghangdoi[MAXN], trongblossom[MAXN];
int hangdoi[MAXN], dau = 0, cuoi = 0;
int sodinh, socanh;
void dayvaohangdoi(int x) {
```

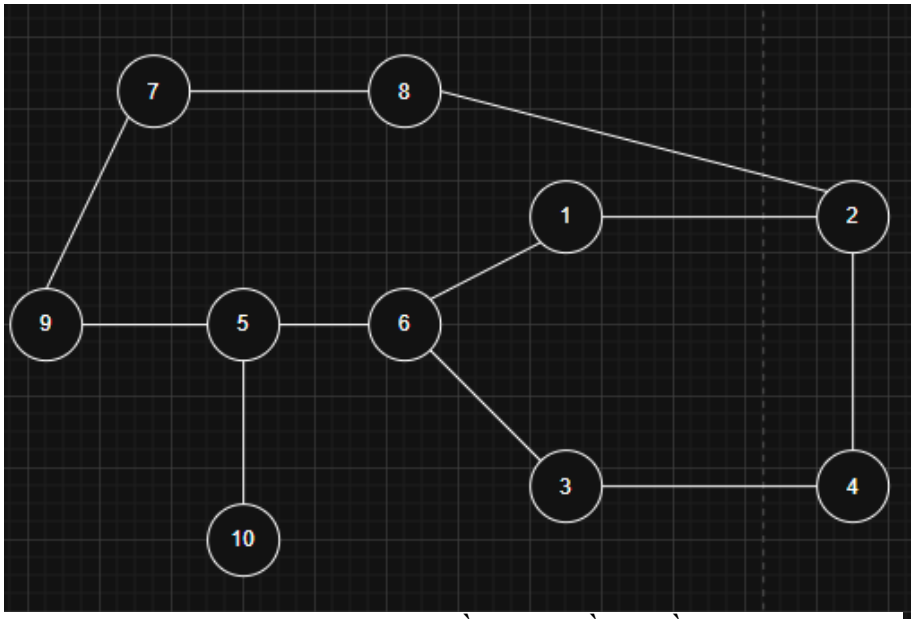
```
    hangdoi[cuoi++] = x;
    tronghangdoi[x] = true;
}
int layrahangdoi() {
    return hangdoi[dau++];
}
int timlca(int a, int b) {
    bool datham[MAXN] = {false};
    while (true) {
        a = goc[a];
        datham[a] = true;
        if (ghep[a] == -1) break;
        a = cha[ghep[a]];
    }
    while (true) {
        b = goc[b];
        if (datham[b]) return b;
        if (ghep[b] == -1) break;
        b = cha[ghep[b]];
    }
    return -1;
}
void danhdaublossom(int u, int v, int lca) {
    while (goc[u] != lca) {
        trongblossom[goc[u]] = trongblossom[goc[ghep[u]]] = true;
        cha[u] = v;
        v = ghep[u];
```

```
        u = cha[v];
    }
}
void rutgonblossom(int u, int v) {
    int lca = timlca(u, v);
    for (int i = 1; i <= sodinh; ++i) trongblossom[i] = false;
    danhdaublossom(u, v, lca);
    danhdaublossom(v, u, lca);
    for (int i = 1; i <= sodinh; ++i) {
        if (trongblossom[goc[i]]) {
            goc[i] = lca;
            if (!tronghangdoi[i]) dayvaohangdoi(i);
        }
    }
}
bool bfs(int gocr) {
    for (int i = 1; i <= sodinh; ++i) {
        cha[i] = -1;
        tronghangdoi[i] = false;
        goc[i] = i;
    }
    dau = cuoi = 0;
    dayvaohangdoi(gocr);
    while (dau < cuoi) {
        int u = layrahangdoi();
        for (int v = 1; v <= sodinh; ++v) {
            if (ke[u][v] == 0 || goc[u] == goc[v] || ghep[u] == v) continue;
```

```
            if (v == gocr || (ghep[v] != -1 && cha[ghep[v]] != -1)) {
                rutgonblossom(u, v);
            } else if (cha[v] == -1) {
                cha[v] = u;
                if (ghep[v] == -1) {
                    while (v != -1) {
                        int pv = cha[v], mv = ghep[pv];
                        ghep[v] = pv;
                        ghep[pv] = v;
                        v = mv;
                    }
                    return true;
                } else {
                    cha[ghep[v]] = v;
                    dayvaohangdoi(ghep[v]);
                }
            }
        }
    }
    return false;
}
int edmonds() {
    for (int i = 1; i <= sodinh; ++i) ghep[i] = -1;
    int dem = 0;
    for (int i = 1; i <= sodinh; ++i) {
        if (ghep[i] == -1) {
            if (bfs(i)) dem++;
```

```
        }
    }
    return dem;
}
int main() {
    cout << "Nhap so dinh va so canh: ";
    cin >> sodinh >> socanh;
    for (int i = 0; i < socanh; ++i) {
        int u, v;
        cin >> u >> v;
        ke[u][v] = ke[v][u] = 1;
    }
    int ketqua = edmonds();
    cout << "So cap ghep cuc dai: " << ketqua << endl;
    for (int i = 1; i <= sodinh; ++i) {
        if (ghep[i] != -1 && i < ghep[i])
            cout << i << " - " << ghep[i] << endl;
    }
    return 0;
}
```

3.2 Kết quả thử



Hình 7: Minh họa đầu vào bằng đồ thị

```
Nhap so dinh va so canh: 10 11
1 2
1 6
2 4
2 8
3 4
3 6
5 6
5 9
5 10
7 8
7 9
So cap ghep cuc dai: 5
1 - 6
2 - 8
3 - 4
5 - 10
7 - 9
```

Hình 8: Kết quả sau khi chạy chương trình

IV/ KẾT LUẬN

Thuật toán Edmonds' Blossom là một giải pháp hiệu quả cho bài toán tìm ghép cặp cực đại trong đồ thị tổng quát. Với cơ chế phát hiện và rút gọn các chu trình lẻ, thuật toán cho phép xử lý chính xác các trường hợp phức tạp mà các phương pháp thông thường không đáp ứng được.

Việc cài đặt và áp dụng thuật toán cho thấy tính ứng dụng cao trong các lĩnh vực như phân công công việc, ghép cặp đề tài - sinh viên, hệ thống gợi ý. Kết quả thực nghiệm đảm bảo mỗi cặp ghép được xác định là tối ưu và không trùng lặp.

Tuy nhiên, thuật toán còn tồn tại một số hạn chế như cấu trúc cài đặt phức tạp, chưa hỗ trợ đồ thị có trọng số và chưa sử dụng được nhiều thư viện của chương trình C++.

Trong tương lai, thuật toán có thể được mở rộng bằng cách cải tiến cấu trúc dữ liệu, kết hợp thư viện đồ thị chuyên dụng hoặc xây dựng giao diện trực quan nhằm mô phỏng và áp dụng vào các hệ thống thực tế quy mô lớn.

TÀI LIỆU THAM KHẢO

Brilliant.org. (n.d.). *Matching Algorithms*. Được truy lục từ Brilliant.org: https://brilliant.org/wiki/matching-algorithms/

contributors, W. (n.d.). *Blossom algorithm*. Được truy lục từ Wikipedia: https://en.wikipedia.org/wiki/Blossom_algorithm

Edmonds, J. (1991). A glimpse of heaven. Trong *History of Mathematical Programming – A Collection of Personal Reminiscences* (trang 32–54). Amsterdam: CWI and North-Holland.

Wiki, V. (n.d.). *Lý thuyết đồ thị - Bài toán Matching cực đại*. Được truy lục từ VNOI Wiki: https://wiki.vnoi.info/algo/graph-theory/max-matching