HỘI THẢO CÁC TRƯỜNG THPT CHUYÊN KHU VỰC DUYÊN HẢI VÀ ĐỒNG BẰNG BẮC BỘ – NĂM 2021

---o0o---

CHUYÊN ĐỀ

SẮP XẾP TOPO TRÊN ĐỒ THỊ CÓ HƯỚNG KHÔNG CHU TRÌNH (DAG)

Tháng 9/2021

MỤC LỤC

Trang

PHẦN 1: MỞ ĐẦU

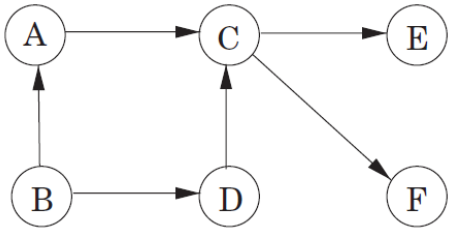
Trong các kì thi học sinh giỏi môn tin học những năm gần đây ngày càng được nâng cao về chất lượng, nội dung thi có tính tổng hợp kiến thức rất cao, kết hợp nhiều kĩ thuật xử lý. Việc cung cấp kiến thức nền tảng và kĩ năng lập trình làm cơ sở cho học sinh là rất cần thiết. Các bài toán vận dụng lý thuyết đồ thị là một trong những nội dung quan trọng và có nhiều thuật toán liên quan. Trong phạm vi chuyên đề, tôi xin trình bày một phần nhỏ của nội dung này đó là **"Sắp xếp Topo trên đồ thị có hướng không chu trình (DAG)"**. Đây là phần kiến thức giúp học sinh bắt đầu tiếp cận các dạng bài toán kết hợp quy hoạch động trên đồ thị. Hệ thống bài tập được tìm hiểu, sưu tầm từ các tài liệu tin học và các trang web lập trình trực tuyến. Tôi hy vọng sẽ hỗ trợ một phần nhỏ cho quý thầy cô và học sinh trong công tác giảng dạy và học tập.

PHẦN 2: NỘI DUNG

1. LÝ THUYẾT

1.1. DAG (Directed Acyclic Graph)

Một đồ thị có hướng G=(V,E) được gọi là một đồ thị có hướng không chu trình (DAG) là đồ thị không tồn tại đường đi khép kín. Cũng có thể hình dung đây là đồ thị mà số lượng đỉnh trong tất cả các thành phần liên thông mạnh đều bằng 1.

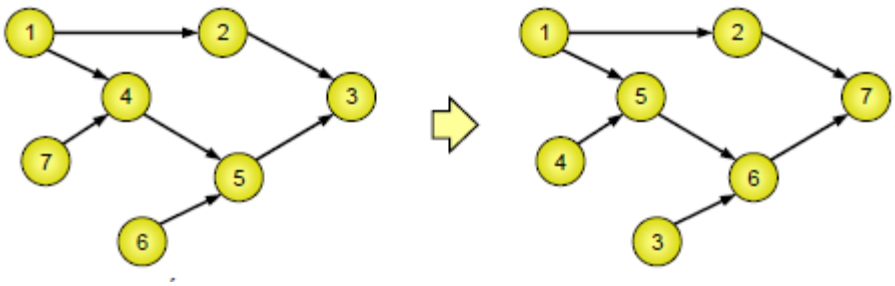


DAG cho phép mô hình hiệu quả các bài toán liên quan đến quan hệ nhân quả, phân cấp, phụ thuộc thời gian.

1.2. Sắp xếp Topo trên DAG

Trong khoa học máy tính, thứ tự topo của một đồ thị có hướng là một thứ tự sắp xếp của các đỉnh sao cho với mọi cung từ u đến v trong đồ thị, u luôn nằm trước v. Thuật toán để tìm thứ tự topo gọi là thuật toán sắp xếp topo. Thứ tự topo tồn tại khi và chỉ khi đồ thị là một DAG. DAG luôn có ít nhất một thứ tự topo.

Bài toán: Cho đồ thị có hướng không chu trình (DAG). Tìm cách sắp xếp thứ tự các đỉnh sao cho với mọi cung từ u đến v trong đồ thị, u luôn nằm trước v.



Thuật toán: Có hai cách để xây dựng một sắp xếp topo trên DAG:

- Cách 1: Dựa vào một tiêu chí tự nhiên mà nếu sắp xếp tăng/giảm theo tiêu chí này thì hiển nhiên ta có một sắp xếp topo.

+ Ý tưởng: Thêm ngẫu nhiên một đỉnh u không có cung vào ở trạng thái hiện tại của đồ thị vào trong mảng topo, bỏ tất cả các cung đi ra từ u đi và lặp lại như bước đầu cho tới khi tập đỉnh của đồ thị hiện tại rỗng. Nếu đồ thị tồn tại trạng thái không tồn tại một đỉnh không có cung vào mà tập cạnh khi đó khác rỗng, hoặc số lượng đỉnh trong mảng topo khác |V| thì chứng tỏ đồ thị có chu trình, và không phải một DAG.

+ Cài đặt:

```
void toposort()
{
   for(int u=1; u<=n; ++u) if(deg[u]==0) q[++R]=u;
   while(R)
   {
      int u=q[R--];
      topo[++sl]=u;
      for(int i=0; i<g[u].size(); ++i)
      {
         int v=g[u][i];
         --deg[v];
         if(deg[v]==0) q[++R]=v;
      }
   }
}
```

- Cách 2: Xây dựng dựa vào thuật toán Tarjan tìm thành phần liên thông mạnh. Khi đồ thị không có chu trình thì các thành phần liên thông mạnh đều có số lượng đỉnh bằng 1. Do vậy trong trường hợp này ta chỉ cần liệt kê các đỉnh theo thứ tự của phép duyệt DFS. Đây là cách tổng quát áp dụng cho mọi trường hợp.

+ Ý tưởng: Sau khi duyệt DFS trên đồ thị, ta sẽ có một mảng kết quả lưu các đỉnh theo thứ tự thăm xong trước tới sau, mảng này chính là trật tự topo.

+ Cài đặt:

```
void DFS(int u)
{
   dd[u]=1;
   for(int i=0; i<g[u].size(); ++i)
   {
      int v=g[u][i];
      if(dd[v]==0)
```

```
        DFS(v);
    }
    topo[++sl]=u;
}
```

Độ phức tạp thuật toán: O(|V|+|E|)

2. BÀI TẬP VẬN DỤNG

2.1. ĐƯỜNG ĐI DÀI NHẤT 1

Đề bài

Cho đồ thị có hướng, không chu trình G gồm n đỉnh và m cạnh. Các đỉnh được đánh thứ tự từ 1 đến n.

Yêu cầu: Tìm đường đi qua nhiều cạnh nhất trong G.

Dữ liệu: Vào từ file LPATH1.INP bao gồm:

- Dòng đầu tiên gồm 2 số nguyên n, m ($1 \le n, m \le 10^5$).
- m dòng tiếp theo, mỗi dòng ghi 2 số nguyên dương u, v thể hiện có một cung một chiều nối từ đỉnh u đến đỉnh v ($1 \le u,v \le n$).

Kết quả: Ghi ra file văn bản LPATH1.OUT gồm 1 số nguyên duy nhất là kết quả bài toán.

Ví dụ:

LPATH1.INP	LPATH1.OUT
5 8 5 3 2 3 2 4 5 2 5 1 1 4 4 3 1 3	3

Thuật toán

DFS từ 1 để đánh lại chỉ số thứ tự các đỉnh của đồ thị theo topo. Gọi f[u] là đường đi dài nhất từ s tới u. Nếu có một cung đi từ u tới v, thì có:

$$F[v] = Max (f[v], f[u] + 1)$$

Chương trình

```
#include<bits/stdc++.h>
```

```
using namespace std;
const int N = 1e5+5;
vector<int> adj[N];
vector<bool> d(N);
int k[N];
void dfs(int n,vector<int> &s){
   for(auto u:adj[n]){
      if(!d[u])
         dfs(u,s);
   }
   d[n]=1;
   s.push_back(n);
}
int main(){
   freopen("Lpath1.inp","r",stdin);
   freopen("Lpath1.out","w",stdout);
   int n,m;
   cin>>n>>m;
   for(int i=1;i<=m;i++){
     int x,y;
     cin>>x>>y;
     adj[x].push_back(y);
   }
   vector<int> s;
   for(int i=1;i<=n;i++){
      if(!d[i]){
         dfs(i,s);
      }
   }
   reverse(s.begin(),s.end());
   int res=0;
   for(int i=0;i<s.size();i++){
      for(auto u:adj[s[i]]){
         k[u] = max(k[u],k[s[i]]+1);
      }
      res= max(res,k[s[i]]);
   }
   cout<<res;
   return 0;
}
```

Link test

https://drive.google.com/drive/folders/1Le34Hlo3o5TfDs8QGh2CJ-Db8UmV_DPM?usp=sharing

2.2. ĐƯỜNG ĐI DÀI NHẤT 2

Đề bài

Cho đồ thị có hướng, không có chu trình, có trọng số và 2 đỉnh s,t. Tìm đường đi ngắn nhất từ s tới t.

Dữ liệu: Vào file LPATH2.INP gồm có:

- Dòng đầu tiên ghi hai số nguyên dương n, m ($1 \le n \le 3.10^5$, $1 \le m \le 5.10^5$) lần lượt là số đỉnh và số cạnh của đồ thị. Các đỉnh đánh số từ 1 đến n

- Dòng thứ hai ghi hai số nguyên dương s, t ($1 \le s, t \le n, s \ne t$)

- m dòng tiếp theo, mỗi dòng ghi 3 số nguyên dương u, v, c ($1 \le u, v \le n, u \ne v, |c| \le 10^3$) thể hiện có một cung một chiều nối từ đỉnh u đến đỉnh v có trọng số là c.

- Dữ liệu đảm bảo rằng đồ thị không có chu trình.

Kết quả: Ghi file LPATH2.OUT gồm có gồm 1 số duy nhất là độ dài của đường đi ngắn nhất từ s đến t. Nếu không tồn tại đường đi như vậy thì ghi -1

Ví dụ:

LPATH2.INP	LPATH2.OUT
5 6 1 5 1 4 5 4 5 -4 1 2 1 2 3 2 3 5 3 1 3 -7	6

Thuật toán

DFS từ s để đánh lại chỉ số thứ tự các đỉnh của đồ thị theo topo. Gọi f[u] là đường đi ngắn nhất từ s tới u. Khi đó f[s]=0, f[u]=+oo với $\forall u \ne s$. Nếu có một cung đi từ u tới v, thì có:

$$F[v] = \text{Max}(f[v], f[u] + c[u,v])$$

Chương trình

```
#include <bits/stdc++.h>
#define maxn 300005
#define oo 1000000001
using namespace std;
```

```
typedef pair<int,int> II;
int n, m, s, t;
vector<II> g[maxn];
int topo[maxn], sl, dd[maxn];
int kc[maxn];
void dfs(int u)
{
    dd[u]=1;
    for(int i=0;i<g[u].size();i++)
    {
        int v=g[u][i].first;
        if (dd[v]==0) dfs(v);
    }
    topo[--sl]=u;
}
int main()
{
    freopen("Lpath2.inp","r",stdin);
   freopen("Lpath2.out","w",stdout);
    cin>>n>>m;
    cin>>s>>t;
    for(int i=1; i<=m; i++)
    {
        int u,v,w;
        cin>>u>>v>>w;
        g[u].push_back(II(v,w));
    }
    memset(dd,0,sizeof(dd));
    sl=n+1;
    for(int i=1; i<=n; i++)
        if (dd[i]==0) dfs(i);
    for(int i=1; i<=n; i++) kc[i]=-oo;
    for(int i=1; i<=n; i++)
    {
        int u=topo[i];
        if (u==s) kc[u]=0;
        for(int i=0;i<g[u].size();i++)
        {
            int v=g[u][i].first, L=g[u][i].second;
            kc[v]=max(kc[v],kc[u]+L);
        }
    }
    if (kc[t]==-oo) cout<<-1;
    else cout<<kc[t];
```

```
    return 0;
}
```

Link test

https://drive.google.com/drive/folders/1fBzs6SMAOJCzqJ7cK7DMJeck2k03TxBt?usp=sharing

Lưu ý: Áp dụng tương tự ta có thể giải bài toán tìm đường đi ngắn nhất

2.3. TÌM TỔNG SỐ ĐƯỜNG ĐI

Đề bài

Cho đồ thị có hướng, không có chu trình, có trọng số và 2 đỉnh s,t. Tìm tổng số đường đi từ s tới t.

Dữ liệu: Vào file SPATH.INP gồm có:

- Dòng đầu tiên ghi hai số nguyên dương n, m ($1 \le n \le 3.10^5$, $1 \le m \le 5.10^5$) lần lượt là số đỉnh và số cạnh của đồ thị. Các đỉnh đánh số từ 1 đến n

- Dòng thứ hai ghi hai số nguyên dương s, t ($1 \le s, t \le n, s \ne t$)

- m dòng tiếp theo, mỗi dòng ghi 2 số nguyên dương u, v ($1 \le u, v \le n, u \ne v, |c| \le 10^3$) thể hiện có một cung một chiều nối từ đỉnh u đến đỉnh v.

Kết quả: Ghi file SPATH.OUT gồm có gồm 1 số duy nhất là số lượng đường đi từ s đến t. Vì con số này có thể rất lớn nên chỉ cần lấy phần dư của chúng khi chia cho 10^9+7

Ví dụ:

SPATH.INP	SPATH.OUT
5 6 1 5 1 4 4 5 1 2 2 3 3 5 1 3	3

Thuật toán

DFS từ s để đánh lại chỉ số thứ tự các đỉnh của đồ thị theo topo. Nếu có một cung đi từ u tới v, thì có: F[u] = f[u]+f[v]

Chương trình

```
#include <bits/stdc++.h>
#define maxn 300005
#define oo 1000000001
#define modulo 1000000007
using namespace std;
typedef pair<int,int> II;
int n, m, s, t;
vector<int> g[maxn];
int topo[maxn], sl, d[maxn];
int f[maxn];
void dfs(int u)
{
   d[u]=1;
   for(auto v:g[u])
   {
      if (d[v]==0) dfs(v);
   }
   topo[--sl]=u;
}
int main()
{
   freopen("Spath.inp","r",stdin);
   freopen("Spath.out","w",stdout);
   cin>>n>>m;
   cin>>s>>t;
   for(int i=1; i<=m; i++)
   {
      int u,v;
      cin>>u>>v;
      g[u].push_back(v);

   }
   memset(d,0,sizeof(d));
   sl=n+1;
   for(int i=1; i<=n; i++)
      if (d[i]==0) dfs(i);

   for(int i=1; i<=n; i++) f[i]=0;
   for(int i=1; i<=n; i++)
   {
      int u=topo[i];
      if (u==s) f[u]=1;
      for(auto v:g[u])
```

```
        {
            f[v]=(f[v]+f[u]) % modulo;
        }
    }
    cout<<f[t];
    return 0;
}
```

Link test

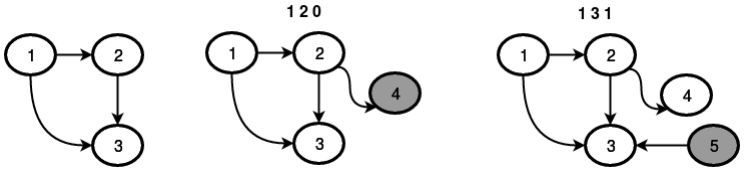
https://drive.google.com/drive/folders/1jCJIa8ovkgj2kAXNRtjNdUD8V4Ocqyu-?usp=sharing

2.4. VÙNG ĐẤT MỚI

Đề bài

Một đất nước có n các thành phố được nối với nhau bằng m con đường một chiều. Các thành phố được đánh số từ 1 đến n. Gần đây chính phủ quyết định xây dựng một số thành phố mới. Để thực hiện cần có q công việc. Mỗi công việc được thể hiện một trong 2 hình thức sau:

- *1 x d*: thành phố n+1 được xây dựng và kết nối với thành phố x. Nếu d =0, chiều con đường mới đi từ thành phố x đến thành phố n+1; nếu d=1 thì có chiều ngược lại



- *2 x y*: Ghi ‘Yes’ nếu tồn tại đường đi từ x đến y. Ghi ‘No’ cho trường hợp ngược lại

Yêu cầu: Xác định có tồn tại đường đi từ x đến y hay không trong công việc ở dạng 2

Dữ liệu: file EPATH.INP

- Dòng đầu tiên gồm 2 số nguyên n, m ($1 \le n,m \le 5.10^4$).

- m dòng tiếp theo, mỗi dòng ghi 2 số nguyên dương u, v thể hiện có một cung một chiều nối từ đỉnh u đến đỉnh v ($1 \le u,v \le n$)

- Dòng tiếp theo ghi số nguyên dương q ($1 \le q \le 10^5$)

- q dòng còn lại, mỗi dòng cho biết hình thức của mỗi công việc ($d \in \{0,1\}$)

Kết quả: file EPATH.OUT cho biết kết quả của các công việc ở dạng 2.

Ví dụ:

EPATH.INP	EPATH.OUT
4 4 1 2 1 3 2 4 3 4 5 1 2 0 2 3 5 2 1 5 1 1 1 2 6 4	NO YES YES

Thuật toán

Đọc hết các truy vấn, thêm tất cả các cạnh trong truy vấn vào đồ thị đã cho.

Tạo đồ thị mới bằng cách gộp tất cả những cạnh nằm trong cùng một thành phần liên thông lại với nhau. Đồ thị mới có dạng DAG.

Với mỗi đỉnh u trong DAG, ta tính bitset dp[u] với ý nghĩa dp[u][v]=1 nếu có đường đi từ đỉnh u đến đỉnh v.

Với mỗi truy vấn x y, tìm 2 đỉnh tương ứng u v trong DAG, xuất Yes nếu dp[u][v]=1.

Chương trình

```
#include <bits/stdc++.h>
#define pb push_back
#define mp make_pair
using namespace std;
const int MAXN = (int)5e5+228;
bitset <50005> dp[50005];
bool have[MAXN];
int n, m;
```

```
vector < int > g[MAXN], gr[MAXN];
int type[MAXN], from[MAXN], go[MAXN], sz, wh[MAXN];
vector < int > ts;
bool used[MAXN];
vector < int > cmps[MAXN];
vector < pair < int, int > > edge;
void dfs(int x) {
   used[x] = 1;
   for(auto &to : g[x]) if(!used[to])
      dfs(to);
   ts.pb(x);
}
void dfs1(int x) {
   used[x] = 1;
   wh[x] = sz;
   cmps[sz].pb(x);
   for(auto &to : gr[x]) if(!used[to])
      dfs1(to);
}
bool res[MAXN];
int main() {
   freopen("Epath.inp", "r", stdin);
   freopen("Epath.out", "w", stdout);
   cin >> n >> m;
   for(int i=1;i<=m;++i) {
      int x, y; cin >> x >> y;
      edge.pb(mp(x,y));
      g[x].pb(y);
      gr[y].pb(x);
   }
   int q; cin >> q;
   for(int i=1;i<=q;++i) {
      cin >> type[i] >> from[i] >> go[i];
      if(type[i] == 1) {
         n++;
         if(go[i]) {
            edge.pb(mp(n,from[i]));
            g[n].pb(from[i]);
            gr[from[i]].pb(n);
         } else {
            edge.pb(mp(from[i],n));
            g[from[i]].pb(n);
            gr[n].pb(from[i]);
         }
```

```
        }
    }
    for(int i=1;i<=n;++i) if(!used[i]) dfs(i);
    reverse(ts.begin(),ts.end());
    memset(used,0,sizeof used);
    for(auto &to : ts) {
        if(!used[to]) {
            sz++;
            dfs1(to);
        }
    }
    for(int i=1;i<=n;++i) g[i].clear();
    for(auto &to : edge) {
        if(wh[to.first] != wh[to.second]) g[wh[to.first]].pb(wh[to.second]);
    }
    memset(used,0,sizeof used);
    for(int i=1;i<=sz;++i) {
        if(!used[i]) {
            dfs(i);
        }
    }
    for(auto &to : ts) {
        dp[to].set(to,1);
        for(auto &it : g[to]) dp[to] |= dp[it];
    }
    for(int i=q;i>=1;--i) {
        if(type[i] == 1) {
            n--;
        } else {
            if(go[i] > n || from[i] > n) continue;
            res[i] = dp[wh[from[i]]].test(wh[go[i]]);
        }
    }
    for(int i=1;i<=q;++i) {
        if(type[i] == 1) continue;
        if(res[i]) cout << "Yes\n";
        else cout << "No\n";
    }
    return 0;
}
```

Link test

https://drive.google.com/drive/folders/1mAxbuTraCx9AZhfPy4xdT1erTFmzr_U_?usp=sharing

2.5. CẮT CÂY

Đề bài

Cho một đồ thị dạng cây gồm n đỉnh đánh số từ 1 đến n, đỉnh i có trọng số a_i. Cho k lần cắt cây (mỗi lần xóa một cạnh hiện đang có) ta sẽ có một rừng gồm $k+1$ cây. Trọng số của một cây được định nghĩa bằng tổng trọng số các đỉnh có trong cây đó.

Bài toán yêu cầu hãy tìm cách cắt cây sao cho trọng số lớn nhất của các cây sinh ra là nhỏ nhất

Dữ liệu:

- Dòng đầu tiên gồm hai số n, k $(1 \le k < n \le 5.10^5)$
- Dòng thứ hai ghi n số $a_1, a_2, ..., a_n$ là trọng số các đỉnh $(0 \le a_i \le 10^3)$
- $n-1$ dòng tiếp theo, mỗi dòng gồm hai số u, v mô tả một cạnh của cây

Kết quả: Một số nguyên duy nhất là trọng số của cây có trọng số lớn nhất trong phương án tối ưu

Ví dụ:

TRCUT.INP	TRCUT.OUT
8 2 7 4 3 8 5 7 5 4 2 1 3 1 4 3 5 2 6 1 7 6 8 1	20

Thuật toán

Giả sử trọng số lớn nhất của một mảnh không vượt quá M. Mỗi khi có một cây con với trọng số lớn hơn M ta tiến hành cắt cạnh trên cây con này từ con có trọng số lớn nhất bằng tìm kiếm nhị phân.

Chương trình

```
#include <bits/stdc++.h>
#define M 500005
using namespace std;
```

```
int n, k;
int cost[M];
vector<int> a[M];
int dp[M];
priority_queue<int, vector<int>> h;
int cal(int u, int par, int maxi){
    int cnt_cut = 0;
    dp[u] = cost[u];
    for(int v : a[u]){
        if(v == par)
            continue;
        cnt_cut += cal(v, u, maxi);
        dp[u] += dp[v];
    }
    for(int v : a[u]){
        if(v == par)
            continue;
        h.push(dp[v]);
    }
    while(dp[u] > maxi){
        ++cnt_cut;
        dp[u] -= h.top();
        h.pop();
    }
    while(!h.empty())
        h.pop();
    return cnt_cut;
}
int cnp(int l, int r){
    int i = l, j = r, tam = -1;
    while(i <= j){
        int mid = (i + j) / 2;
        int ta = cal(1, 0, mid);
        if(ta <= k){
            tam = mid;
            j = mid - 1;
        }
        else
            i = mid + 1;
    }
    return tam;
}
int main(){
    freopen("TRCUT.inp", "r", stdin);
```

```
    freopen("TRCUT.out", "w", stdout);
    cin >> n >> k;
    for(int i = 1; i <= n; ++i)
        cin >> cost[i];
    for(int i = 1; i < n; ++i){
        int u, v;
        cin >> u >> v;
        a[u].push_back(v);
        a[v].push_back(u);
    }
    int maxi = 0;
    for(int i = 1; i <= n; ++i)
        maxi = max(maxi, cost[i]);
    cout << cnp(maxi, 1000000000);
    return 0;
}
```

Link test

https://drive.google.com/drive/folders/19TyHXHeojTMV3ot_J_AVGqYAV0VYBY_0?usp=sharing

3. BÀI TẬP TỰ LUYỆN

3.1. https://www.spoj.com/problems/DAGCNT/

3.2. https://www.spoj.com/problems/TOPOSORT/

3.3. https://codeforces.cc/gym/101102/problem/K

3.4. https://codeforces.com/problemset/problem/770/C

3.5. https://codeforces.com/contest/919/problem/D

PHẦN III: KẾT LUẬN

Sắp xếp topo là một trong những bài toán có tính ứng dụng cao cả trong Tin học lẫn Toán học và đời sống thường ngày. Việc chỉ ra một sắp xếp topo trên DAG là điều kiện tiên quyết cho các bài toán qui hoạch động trên loại đồ thị này. Đây cũng là một dạng bài giúp học sinh tiếp cận quy hoạc động trong đồ thị. Lớp bài toán về DAG rất phong phú và còn nhiều thú vị. Tuy nhiên do kinh nghiệm còn hạn chế nên chuyên đề chỉ dừng lại một số bài toán vận dụng cơ bản để tiếp cận. Rất mong nhận được đóng góp ý kiến của quý thầy cô để tiếp tục tìm hiểu, nghiên cứu sâu hơn.

Xin chân thành cảm ơn!

TÀI LIỆU THAM KHẢO

[1]. Bài giảng giải thuật và lập trình của thầy Lê Minh Hoàng;

[2]. Các thuật toán trên đồ thị của thầy Lê Thanh Bình ;

[3]. Sách giáo khoa chuyên tin quyển 1,2 ;

[4]. Algorithms, 4th Edition;

[5]. http://www.tailieu.vn;

[6]. http://www.ebook.edu.vn;

[7]. https://cowboycoder.tech

[8]. https://www.spoj.com

[9]. https://codeforces.com