I. CTDL

***List:**

```
typedef int ElementType;
typedef struct
{
    ElementType data[MAXLENGTH];
    int size;
} List;

void makenullList(List *L)
{
    L->size = 0;
}

void pushback(List *L,
ElementType x)
{
    L->data[L->size] = x;
    L->size++;
}

ElementType element_at(List L,
int i)
{
    return L.data[i - 1];
}
```

```
int countlist(List L)
{
    return L.size;
}

void copylist(List *L1, List *L2)
{
    List temp = *L2;
    *L2 = *L1;
    *L1 = temp;
}
```

***QUEUE**

```
typedef struct
{
    int data[MAX_ELEMENTS];
    int front, rear;
} Queue;
void make_null_queue(Queue *Q)
{
    Q->front = 0;
    Q->rear = -1;
}
void push(Queue *Q, int x)
{    Q->rear++;
    Q->data[Q->rear] = x;
}
int top(Queue *Q)
{
    return Q->data[Q->front];
}
void pop(Queue *Q)
{
    Q->front++;
}
int empty(Queue *Q)
{
    return Q->front > Q->rear;
}
```

***Stack**

```
typedef int ElementType;
typedef struct
{
    int data[MAXLENGTH];
    int size;
} Stack;

void makenullStack(Stack *S)
{
    S->size = 0;
}

ElementType top(Stack S)
{
    return S.data[S.size - 1];
}
{
    S->size--;
}

int empty(Stack S)
{
    return S.size == 0;
}

void push(Stack *S, ElementType x)
{
    S->data[S->size] = x;
    S->size++;
}
```

***Ma trận D-D(kề)**

```
typedef struct
{
    int n;
    int A[MAXN][MAXN];
} Graph;
void init_graph(Graph *G, int n)
{
    int i, j;
    G->n = n;
    for (i = 1; i <= n; i++)
        for (j = 1; j <= n; j++)
            G->A[i][j] = 0;
}
void add_edge(Graph *G, int x,
int y)
{
    G->A[x][y] = 1;
    G->A[y][x] = 1;
}
int adjacent(Graph G, int x, int y)
{

void pop(Stack *S)
    if (G.A[x][y] == 1)
        return 1;
    return 0;
}
List neighbors(Graph G, int x)
{
    List L;
    makenullList(&L);
    int i;
    for (i = 1; i <= G.n; i++)
      if (adjacent(G, x, i) == 1)
            pushback(&L, i);
    return L;
}
int degree(Graph G, int x)
{
    int deg = 0;
    int i;
    for (i = 1; i <= G.n; i++)
        if (G.A[x][i] != 0)
            deg += G.A[x][i];
    return deg;
}
```

***DSCung**

```
typedef struct
{
    int x, y;
    int w;
} Edge;
typedef struct
{
    int n, m;
    Edge edges[MAX_EDGES];
} Graph;
```

```
void init_graph(Graph *G, int n)
{
    G->n = n;
    G->m = 0;
}
void add_edge(Graph *G, int x, int
y, int w)
{
    G->edges[G->m].x = x;
    G->edges[G->m].y = y;
    G->edges[G->m].w = w;
    G->m++;
}
```

***Duyệt rộng**

```
void bfs(Graph *G)
{
  Queue q;
  makenull_queue(&q);
  int mark[MAXN];
  int i;
  for (i = 1; i <= G->n; i++)
     mark[i] = 0;
  for (i = 1; i <= G->n; i++)
   if (mark[i] == 0)
   {
     mark[i]=1;
     enqueue(&q,i);
     printf("%d\n",i);
     while (!empty(&q))
     {
        int i;
        int u = top(&q);
        dequeue(&q);
        List list = neighbor(G, u);
        for (i = 1; i <= list.idx;
i++)
        if (mark[elementAt(&list, i)]
        == 0)
        {
           printf("%d\n",
                  elementAt(&list, i));
          mark[elementAt(&list, i)]=1;
              enqueue(&q,
              elementAt(&list, i));
         }
       }
     }
}
```

***Duyệt sâu (Stack)**

```
void dfs(Graph *G)
{
  int mark[MAXELEMENT];
  int i, j;
  Stack S;
  makenull_stack(&S);
  for (i = 0; i <= G->n; i++)
    mark[i] = 0;
  for (i = 1; i <= G->n; i++)
    if (mark[i] == 0)
    {
       push(&S, i);
       while (!emptystack(&S))
       {
          int u = top(&S);
          pop(&S);
          if (mark[u] == 1)
            continue;
          printf("%d\n", u);
          mark[u] = 1;
          List list;
          makenull_list(&list);
          list = neighbor(G, u);
          for (j = 1; j <=
list.idx; j++)
              push(&S,
elementAt(&list, j));
       }
    }
}
```

***Duyệt sâu (đệ quy)**

```
int mark[MAXELEMENT];

void visit(Graph *G, int u)
{
    if (mark[u] == 1)
        return;
    printf("%d\n", u);
    mark[u] = 1;
    List list = neighbor(G, u);
    int i;
    for (i = 1; i <= list.idx;
i++)
        visit(G, elementAt(&list,
i));
}
```

```
void dfs_dequy(Graph *G)
{
    int i;
    for (i = 0; i < G->n; i++)
        mark[i] = 0;
    for (i = 1; i <= G->n; i++)
        if (mark[i] == 0)
            visit(G, i);
}
```

- Tìm cha thêm int parent[MAXN];
- Riêng Stack thì thay đổi cấu trúc

```
typedef struct{
   int u;
   int parent;
} ELEMENT_TYPE;
```

```
typedef struct{
    int top_idx;
    ELEMENT_TYPE
A[MAXELEMENT];
} Stack;
```

***KTra Chu trình (DSC)**

```
int parent[MAXELEMENT];
int findRoot(int u)
{
    while (u != parent[u])
        u = parent[u];
    return u;
}
int chutrinh(Graph *G)
{
  int i;
  for (i = 1; i <= G->n; i++)
    parent[i] = i;
  for (i = 0; i < G->m; i++)
  {
    int r_u = findRoot(G->edges[i].x);
    int r_v = findRoot(G->edges[i].y);
    if (r_u != r_v)
       parent[r_v] = r_u;
    else
       return 0;
   }
   return 1;
}//0 thì có chu trình
```

***Tìm BPLT**

```
int min(int x, int y)
{
    return (x < y) ? x : y;
}

Stack S;
int on_stack[MAXN];
int num[MAXN], min_num[MAXN], idx;
int count = 0;//Lưu số BPLT
void strongconnect(Graph *G, int x)
{
    int i;
    num[x] = min_num[x] = idx;
    idx++;
    push(&S, x);
    on_stack[x] = 1;
    List list = neighbor(G, x);
    int j;
    for (j = 1; j <= list.idx; j++)
    {
        int y = elementAt(&list, j);
        if (num[y] == -1)
        {
            strongconnect(G, y);
            min_num[x] = min(min_num[x], min_num[y]);
        }
        else if (on_stack[y])
            min_num[x] = min(min_num[x], num[y]);
    }
    if (num[x] == min_num[x])
    {
        count++;
        int w;
        do
        {
            w = top(&S);
            pop(&S);
            on_stack[w] = 0;
        } while (w != x);
    }
}
```

***Dijkstra**

```
int mark[MAXN];
int pi[MAXN];
int p[MAXN];
void Dijkstra(Graph *G, int s)
{
  int i, j, it;
  for (i = 1; i <= G->n; i++)
  {
    pi[i] = INFINITY;
    mark[i] = 0;
  }
  pi[s] = 0;
  p[s] = -1;
  for (it = 1; it < G->n; it++)
  {
    int min_pi = INFINITY;
    for (j = 1; j <= G->n; j++)
      if (mark[j] == 0 && pi[j] <
min_pi)
      {
        min_pi = pi[j];
        i = j;
      }
    if (G->w[i][j] != NOEDGE &&
        mark[j] == 0)
    {
      if (pi[i] + G->w[i][j] <
          pi[j])
      {
        pi[j] = pi[i] + G-
                        >w[i][j];
        p[j] = i;
      }
    }
  }
}
//Từ s tới t
// printf("%d", pi[t]+pi[s]);
```

***BellmanFord**

```
int pi[MAXN];
int p[MAXN];
void BellmaFord(Graph *G, int s)
{
  int j, it;
  int i;
  for (i = 1; i <= G->n; i++)
    pi[i] = INFINITY;
  pi[s] = 0;
  p[s] = -1;
  for (it = 1; it < G->n; it++)
    for (j = 0; j < G->m; j++){
      mark[i] = 1;
      for (j = 1; j <= G->n; j++)
      int x = G->edges[j].x;
      int y = G->edges[j].y;
      int w = G->edges[j].w;
      if (pi[x] + w < pi[y])
      {
        pi[y] = pi[x] + w;
        p[y] = x;
      }
    }
  }
}
```

***Chu trình âm**

```
void checkChuTrinhAm(Graph *G,
int s)
{
    BellmaFord(G, s);
    int j;
    for (j = 0; j < G->m; j++)
    {
        int x = G->edges[j].x;
        int y = G->edges[j].y;
        int w = G->edges[j].w;
        if (pi[x] + w < pi[y])
        {
            printf("YES");
            return;
        }
    }
    printf("NO");
}
```

***Ranking Sort**

```
void topo_sort(Graph *G, List *L)
{
    ranking(G);
    makenullList(L);
    int i, j;
    for (i = 0; i <= k; i++)
        for (j = 1; j <= G->n;
j++)
            if (rank[j] == i)
                pushback(L, j);
}
```

***Ranking**

```
int rank[MAXVERTICLES];
int k = 0;
void ranking(Graph *G)
{
  int d[MAXVERTICLES];
  int x, u;
  for (u = 1; u <= G->n; u++)
  {
    d[u] = 0;
    rank[u] = 0;
  }
  for (x = 1; x <= G->n; x++)
    for (u = 1; u <= G->n; u++)
       if (G->A[x][u] != 0)
           d[u]++;//d[v]++
  List S1, S2;
  makenullList(&S1);
  for (u = 1; u <= G->n; u++)
     if (d[u] == 0)
        pushback(&S1, u);
  int i;
  while (S1.size > 0)
  {
    makenullList(&S2);
    for (i = 1; i <= S1.size; i++){
    //Ngược: (i = S1.size; i >= 1;
    i--)
        int u = element_at(&S1, i);
        rank[u] = k;
        int v;
        for (v = 1; v <= G->n; v++)
          if (G->A[u][v] != 0){
             d[v]--;
             if (d[v] == 0)
               pushback(&S2, v);
           }
      }
      copylist(&S1, &S2);
      k++;
    }
  }
```

***Thi công**

```
void main(){
//Đọc đồ thị
  Graph G;
  int n, u, x, v, j;
  freopen("dt.txt", "r", stdin);
  scanf("%d", &n);
  init_graph(&G, n + 2);
  //Đỉnh alpha = 0
  d[n + 1] = 0;
  for (u = 1; u <= n; u++)
  {
    scanf("%d", &d[u]);
    do
    {
       scanf("%d", &x);
       if (x > 0)
         add_edge(&G, x, u);
    } while (x > 0);
  }
//Thêm cung từ alpha tới đỉnh bậc 0
  for (u = 1; u <= n; u++)
  {
     int deg_neg = 0;
     for (x = 1; x <= n; x++)
       if (G.A[x][u] > 0)
          deg_neg++;
       if (deg_neg == 0)
          add_edge(&G, n + 1, u);
   }
//Thêm cung từ đỉnh cuối tới beta
   for (u = 1; u <= n; u++)
   {
     int deg_pos = 0;
     for (x = 1; x <= n; x++)
       if (G.A[u][x] > 0)
           deg_pos++;
       if (deg_pos == 0)
          add_edge(&G, u, n + 2);
   }
   List L;
   int i;
   //Sắp sếp theo rank lưu vào
List
   topo_sort(&G, &L);
//Tính t[u]
   int t[MAXVERTICLES];
   t[n + 1] = 0;
   for (i = 2; i <= L.size; i++)
   {
     int u = element_at(&L, i);
     t[u] = -1;
     for (x = 1; x <= G.n; x++)
        if (G.A[x][u] > 0)
        t[u] = max(t[u], t[x] +
                          d[x]);
    }
//Tính T[u]
int T[MAXVERTICLES];
T[n + 2] = t[n + 2];
for (i = L.size - 1; i >= 1; i--)
  {
    int u = element_at(&L, i);
    T[u] = INFINITY;
    for (x = 1; x <= G.n; x++)
      if (G.A[u][x] > 0)
        T[u] = min(T[u], T[x] -
                          d[u]);
  }
}

//Tìm đường Gantt
//for (i = 1; i <= n+2; i++)
//        if (t[i] == T[i])
//           printf("%d\n", i);
```

***Kruskal**

```
int parent[MAXN];
int findRoot(int u)
{
  return (parent[u] == u) ? u :
            findRoot(parent[u]);
}
void swap(Edge *e1, Edge *e2)
{
    Edge temp = *e1;
    *e1 = *e2;
    *e2 = temp;
}
int Kruskal(Graph *G, Graph *T)
{
    //Edges Sort
    int i, j;
    for (i = 0; i <= G->m - 1;
i++)
        for (j = i + 1; j <= G->m
- 1; j++)
            if (G->edges[i].w >
G->edges[j].w)
                swap(&(G-
>edges[i]), &(G->edges[j]));
    //Kruskal
    init_graph(T, G->n);
    int u;
    for (u = 1; u <= G->n; u++)
        parent[u] = u;

    int sum = 0;
    int e;
    for (e = 0; e <= (G->m - 1);
e++)
    {
        int u = G->edges[e].x;
        int v = G->edges[e].y;
        int w = G->edges[e].w;
        int root_u = findRoot(u);
        int root_v = findRoot(v);
        if (root_u != root_v)
        {
          add_edge(T, u, v, w);
          parent[root_v] = root_u;
          sum += w;
         }
      }
      return sum;
}
```

***PRIM**

```
int mark[MAXN];
int pi[MAXN];
int p[MAXN];
int Prim(Graph *G, Graph *T){
  init_graph(T, G->n);
  int i, u, v;
  for (i = 1; i <= G->n; i++)
  {
     pi[i] = INFINITY;
     mark[i] = 0;
  }
  pi[1] = 0;
  mark[1] = 1;
  for (v = 1; v <= G->n; v++)
    if (G->w[1][v] != NOEDGE)
    {
      pi[v] = G->w[1][v];
      p[v] = 1;
    }
   int sum = 0;
   for (i = 1; i < G->n; i++)
   {
     int min_dist = INFINITY,
                       min_u;
     for (u = 1; u <= G->n; u++)
       if (mark[u] == 0)
       if (min_dist > pi[u]){
          min_dist = pi[u];
          min_u = u;
       }
       u = min_u;
       mark[u] = 1;
       add_edge(T, p[min_u], min_u,
              min_dist);
       sum += min_dist;
       for (v = 1; v <= G->n; v++)
          if (G->A[u][v] != NOEDGE
                && mark[v] == 0)
              if (pi[v] > G-
                        >w[u][v])
              {
                pi[v] = G->w[u][v];
                p[v] = u;
              }
   }
   return sum;
}
```

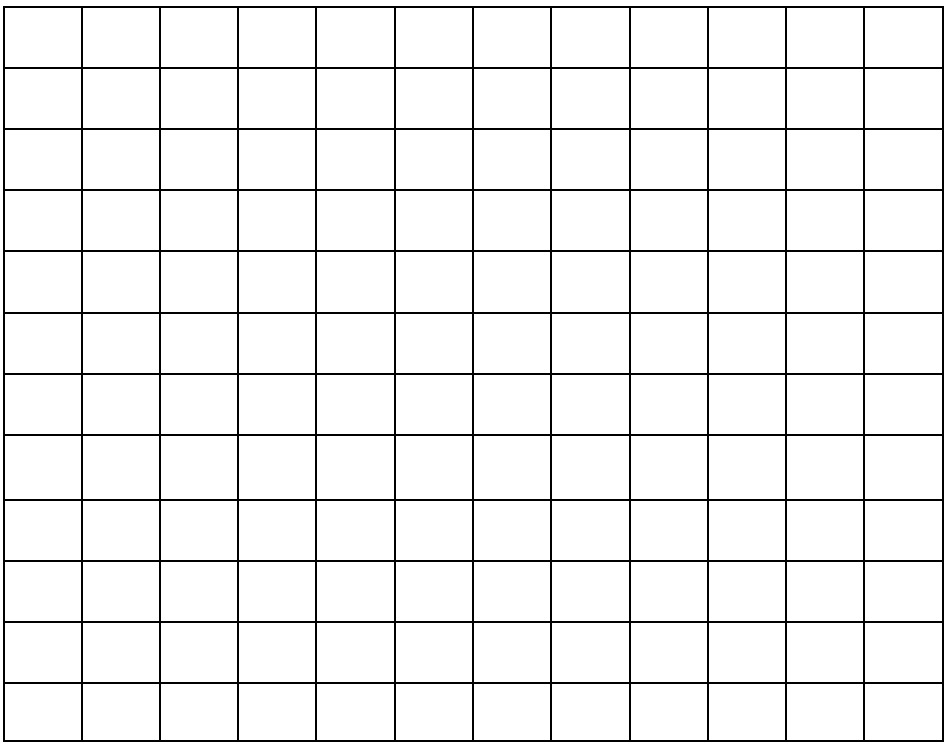
***Luồng**

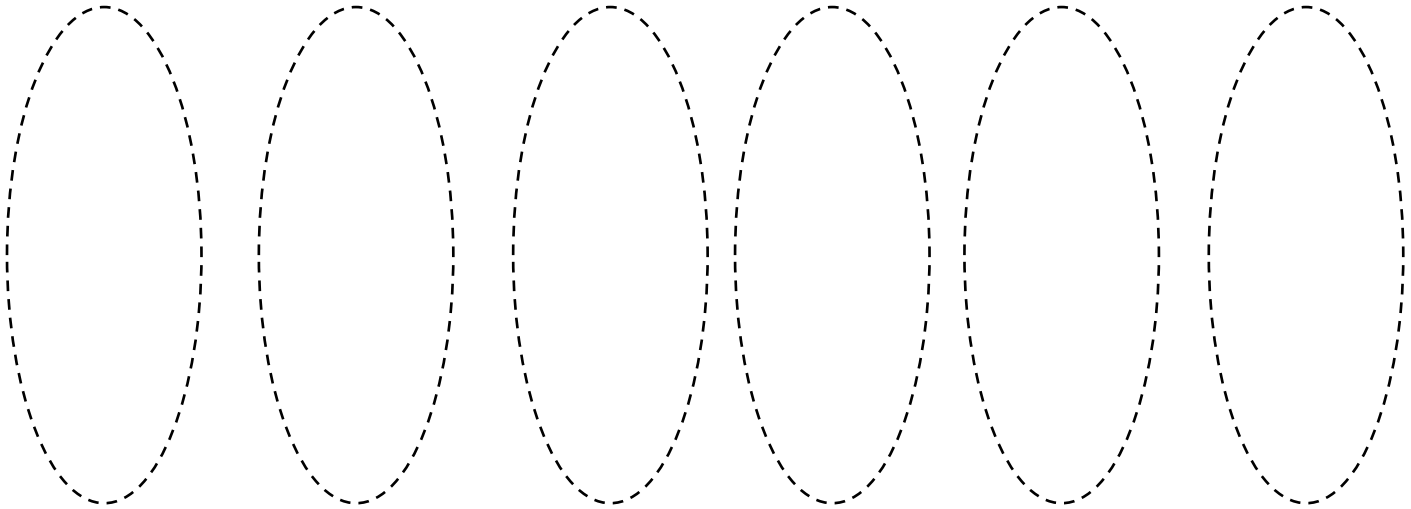
```
//Thiết lập cấu trúc
typedef struct{
    int C[MAXN][MAXN];
    int F[MAXN][MAXN];
    int n;
} Graph;
typedef struct{
    int dir;
    int pre;
    int sigma;
} Label;
Label labels[MAXN];
void init_graph(Graph *G, int n){
    G->n = n;
}
void init_flow(Graph *G){
    int u, v;
    for (u = 1; u <= G->n; u++)
      for (v = 1; v <= G->n; v++)
          G->F[u][v] = 0;
}
void add_edge(Graph *G, int x,
int y, int w){
    G->C[x][y] = w;
}
int min(int x, int y){
    return (x < y) ? x : y;
}
```

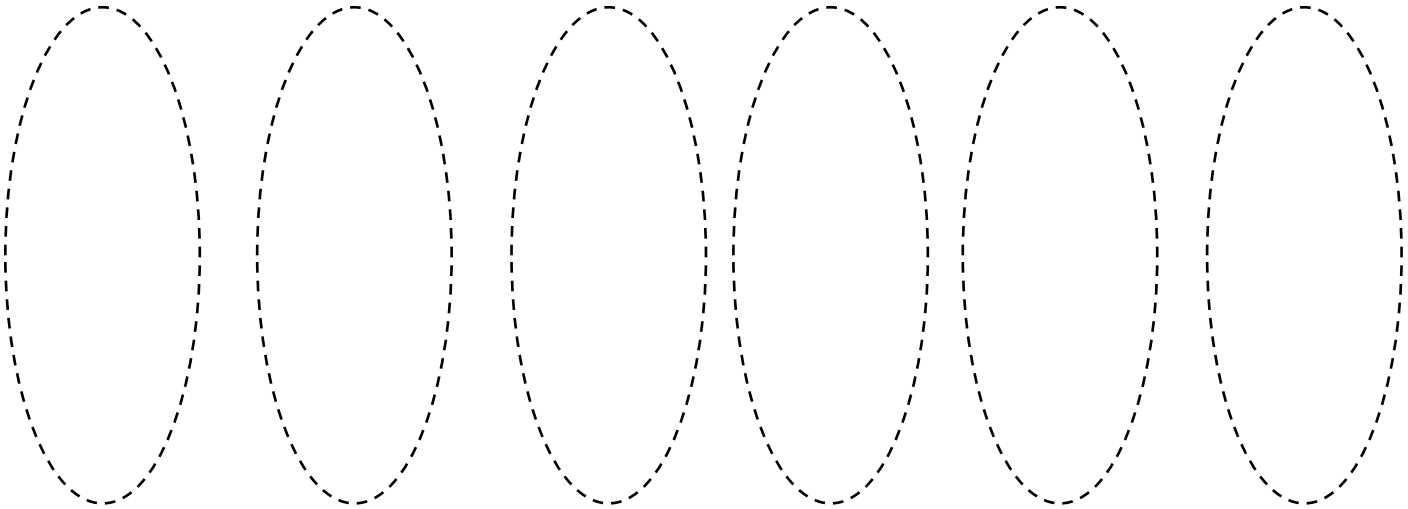
```
int FordFullkerson(Graph *G, int s,
int t){
  init_flow(G);
  Queue Q;
  int sum_flow = 0;
  do{
   //Xoá nhãn rồi gán nhãn cho s
     int u;
     for (u = 1; u <= G->n; u++)
        labels[u].dir = 0;
     labels[s].dir = +1;
     labels[s].pre = s;
     labels[s].sigma = INF;
     make_null_queue(&Q);
     push(&Q, s);
   //Lặp gán nhãn cho các đỉnh
     int found = 0;
     while (!empty(&Q)){
       //Lấy 1 đỉnh trong Queue
       int x = top(&Q);
       pop(&Q);
       int v;
       for (v = 1; v <= G->n; v++){
       //Xét đỉnh kề x, cung thuận
         if (labels[v].dir == 0 &&
         G->C[x][v] != NO_EDGE &&
         G->F[x][v] < G->C[x][v])
         {
           labels[v].dir = +1;
           labels[v].pre = x;
```

```
        labels[v].sigma =
      min(labels[x].sigma, G-
      >C[x][v] - G->F[x][v]);
          push(&Q, v);
        }
    //Xét đỉnh kề x, cung nghịch
   if (labels[v].dir == 0 && G-
       >C[v][x] != NO_EDGE && G-
       >F[v][x] > 0)
   {
     labels[v].dir = -1;
     labels[v].pre = x;
     labels[v].sigma =
      min(labels[x].sigma, G-
      >F[x][v]);
     push(&Q, v);
     }
   }
    //t được gán nhãn thì thoát
while 2 để tăng luồng
   if (labels[t].dir != 0){
     found = 1;
     break;
   }
  }
    //Tăng luồng
  if (found == 1){
    int x = t;
    int sigma = labels[t].sigma;
    sum_flow += sigma;
    while (x != s){
        int u = labels[x].pre;
        if (labels[x].dir > 0){
            //Tăng
          G->F[u][x] += sigma;
        }else{
            /Giảm
          G->F[x][u] -= sigma;
        }
        x = u;
      }
    }else{
       break; //Thoát while 1
     }
   } while (1);
 return sum_flow; //return LCD
}


//In 2 lát cắt
//    printf("X0:");
//    for (u = 1; u <= n; u++){
//        if (labels[u].dir != 0)
//            printf(" %d", u);
//   }
//    printf("\nY0:");
//    for (u = 1; u <= n; u++){
//        if (labels[u].dir == 0)
//            printf(" %d", u);
//    }
```







II. IDEAS

1. **Ma trận đỉnh - đỉnh**
   Bảng 1..n, có cung thì thêm G->A[u][v]=1 or G->A[u][v]=w với trọng số
2. **Danh sách cung**
   Dùng cho Bellman-Ford, Kruskal.
3. **Duyệt rộng**
   Dùng Queue:
   - Khởi tạo mark[u]=0
   - Đưa 1 vào frontier
   - Loop đếm empty queue:
     + Lấy phần tử đầu tiên frontier ra duyệt
     + Lấy neighbor
     + Duyệt đỉnh kề
4. **Duyệt sâu**
   a. **Ngăn xếp:** Như duyệt rông mà dùng *Stack frontier;*
   b. **Đệ quy:**
      - Khởi tao mark[u]=0
      - Nếu mark[u]==0 => Vist đỉnh u (1..n):
        + Duyệt u
        + Lấy neighbor
        + Duyệt đỉnh kề = visit()
5. **Kiểm tra chu trình**
   parent[u] lưu đỉnh cha của u
   Khởi tạo parent[u]=u => mỗi đỉnh là 1 BPLT
   Find root từng cung trong DSC, nếu 2 root trùng nhau sẽ tạo ra chu trình => return 0;
6. **Dijkstra**
   Khởi tạo:
   + p[i] = +oo (1..n);
   + pi[s]=0; p[s]=-1;
   + mark[i]=0 (1..n);
   Lặp n-1 lần:
     + Chọn đỉnh chưa đánh dấu (mark[i]==0) có pi[i] nhỏ nhất;
     + mark[i]=1;
     + for(1->n){
       Nếu pi[i] + G->A[i][j] < pi[j];
       pi[j]=pi[i]+G->A[i][j];
       p[j]=i;
     }

7. **Bellman – Ford**

Khởi tạo:

+ pi[i] = + oo;

+ pi[s]= 0;p[s]=-1;(s: đỉnh bắt đầu duyệt vd:1);

Lặp n-1 lần:

+ for(tất cả các cung)

```
if( pi[u] + w  < pi[v] )
        pi[v]=pi[u] + w;
        p[v]=u;
```

8. **Kiểm tra chu trình âm**

Duyệt qua các cung 1 lần nữa:

If(pi[u] + w < pi[v]) //có chu trình âm

9. **Ranking**

Tính d[u] (G->A[u][v] ==1 : d[v]++);

```
Đưa đỉnh I có d[i] == 0 vào S(List);
K= rank ban đầu mình muốn (0 or 1);
Lặp đến S rỗng (while)
    For(các đỉnh u trong S1) //
        Rank[u]=k;
        For(các đỉnh kề của u)  //v: các đỉnh kề của u;ĐK: G->A[u][v]!=0
            D[v]--;
            Nếu d[v] ==0 thì thêm v vào S2
    K++;
Copy S2 vào S1;
```

- Ngược: Giống như ranking. Nhưng tính d[u] thì đổi d[v]++ thành d[u]++

For(các đỉnh kề của u) thành For(các đỉnh cha của u) //ĐK: G->A[v][u] !=0

10. **Ranking Sort**

```
Ranking(Graph* G,List* L);
Khởi tạo L;
For(i:L->k) //k là bậc lớn nhất ; L là tùy vào lúc đầu khởi tạo k bằng mấy
    For(j:1->n)
        Nếu rank[j]==k thì push vào List
```

11. **Thi công**

12. **Kruskal**

Sắp xếp các cung từ nhỏ đến lớn.

```
Tạo cây T mới vs T->n =G->n;
For(i:1->n) parent[i]=i;
Sum_w=0;
For(e:1->m)//m là số cung của đồ thị G
```

Int u=edge[e].u;

Tương tự vs v,w;

Root_u=findroot(u); root_v=findroot(v);

Nếu root_u != root_v thì thêm cung (u,v) vào cây T và cập nhật lại cha của root_v = root_u; sum_w+=w;

13. Prim

Khởi tạo pi[i]= oo; mark[i]=0; //i:1->n;

Gán pi[1]=0; mark[1]=1;

For(v:1->n) nếu có cung (1,v) thì pi[v]=G->A[1][v];   p[v]=1;

Sum=0;

For(i:1->n) //bé hơn(n-1 lần)

Khởi tạo min_dist=INF, min_u;

For(u:1->n)

Nếu mark[u] ==0 và min_dist > pi[u] thì min_dist=pi[u]; min_u=u;

U=min_u;

Mark[u]=1;

Thêm cung (p[u],u) vào cây T;

Sum+= min_dist;

For(v:1->n)

Nếu có cung(u,v) và mark[v]==0 và pi[v]>G->A[u][v] thì pi[v]=G->A[u][v] ; p[v]=u;