CANTHO UNIVERSITY

Thực hành
KỸ THUẬT NHÁNH CẬN

BÀI TOÁN NGƯỜI GIAO HÀNG (TSP)

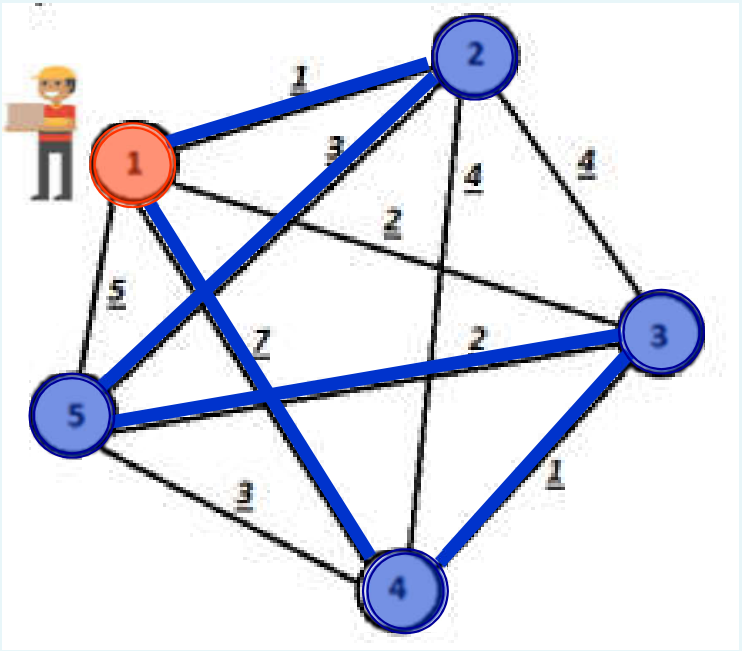
f(X) → MIN

Võ Huỳnh Trâm

CANTHO UNIVERSITY

Bài toán đường đi của người giao hàng (TSP - Travelling Salesman Problem) : Mô tả

- **Bài toán**: Một người giao hàng cần đi giao hàng tại **n** thành phố. Xuất phát từ một thành phố, đi qua các thành phố khác và trở về thành phố ban đầu.
- **Yêu cầu**: Tìm một chu trình sao cho tổng **độ dài** các cạnh là ***nhỏ nhất*** hay một phương án có giá nhỏ nhất ?



CÁC PHƯƠNG PHÁP GIẢI

(1) Vét cạn: Độ phức tạp thời gian là ***hàm mũ***

→ Xét **(n-1)!/2** chu trình.

(2) Tham ăn: - Ưu tiên chọn cạnh nhỏ nhất

- Láng giềng gần nhất

→ ***Không chắc*** tìm được phương án tối ưu

(3) Nhánh cận: Tìm được ***phương án tối ưu*** trong một ***thời gian có thể chấp nhận*** được.

TSP: KỸ THUẬT NHÁNH CẬN – PHÂN NHÁNH (Xây dựng cây TK phương án)

- Nút gốc: xuất phát từ một thành phố (**bậc 0**)
- Nút gốc có n – 1 nút con (**bậc 1**), tương ứng với các khả năng đi ra từ thành phố ở bậc 0.
- Mỗi nút con ở bậc 1 có n – 2 nút con (**bậc 2**), tương ứng với các khả năng đi ra từ thành phố ở bậc 1.
- …
- Đến **bậc n – 1**: đã có n – 1 cạnh, đi đến thành phố cuối cùng, quay về thành phố ban đầu → ***Phương án***

CANTHO UNIVERSITY

TSP: KỸ THUẬT NHÁNH CẬN – TÍNH CẬN (Công thức tính CẬN DƯỚI)

- Nút gốc (**bậc 0**): **CD = n * Độ dài cạnh nhỏ nhất**

- Nút trong (**bậc i**):

TGT = *Tổng độ dài các cạnh đã có*

(từ thành phố xuất phát đến thành phố đang xét)

CD = TGT + (n – i) * Độ dài cạnh nhỏ nhất *kế tiếp*

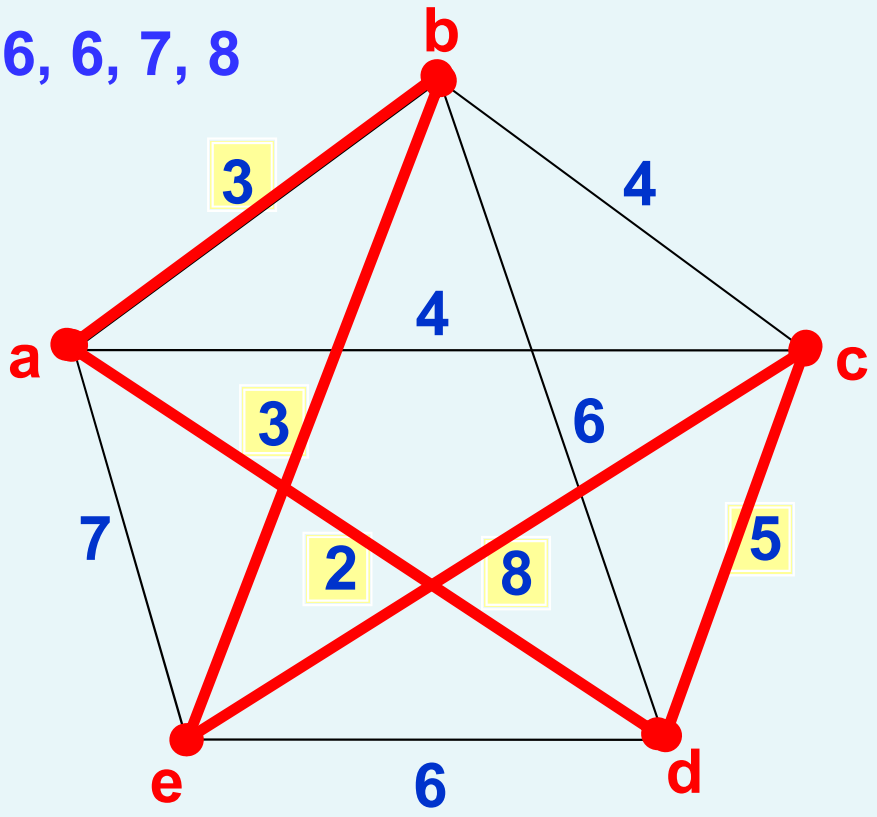
(trong số các cạnh chưa dùng)

Số cạnh còn lại

VÍ DỤ: Bài toán TSP – 5 thành phố

- n = 5 **2, 3, 3, 4, 4, 5, 6, 6, 7, 8**
- Giải bằng phương pháp VÉT CẠN → xét **(n -1)!/2 = 12** chu trình
- Giải bằng phương pháp THAM ĂN (ưu tiên chọn cạnh nhỏ nhất)

→ Chu trình **d a b e c d** có **TGT = 2 + 3 + 3 + 5 + 8 = 21**



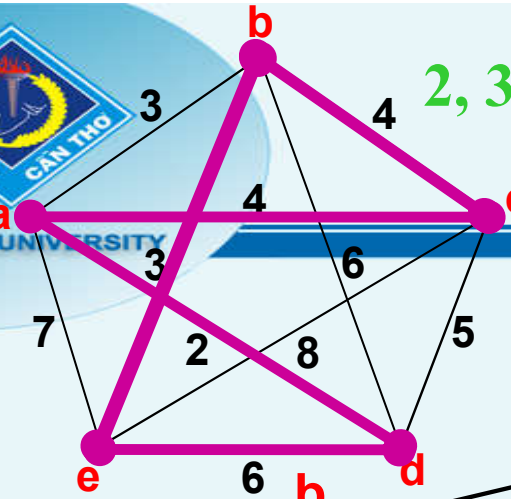
CÔNG THỨC TÍNH CẬN : Bài toán TSP

(1) Nút gốc

- **TGT = 0**
- **CD = n * Độ dài (Cạnh Min)**

(2) Nút trong

- **TGT = TGT (cha) + Độ dài Cạnh**
- **CD = TGT + (n- i) * Độ dài (Cạnh Min kế tiếp)**



2, 3, 3, 4, 4, 5, 6, 6, 7, 8

bậc 0
n-i=5

TGT = 0
CD = 10
d

bậc 1
n-i=4

a	b	c	e
TGT = 2 CD = 14	TGT = 6 CD = 14	TGT = 5 CD = 13	TGT = 6 CD = 14

bậc 2
n-i=3

b	c	e
TGT = 5 CD = 14	TGT = 6 CD = 15	TGT = 9 CD = 18

bậc 3
n-i=2

c	e	b	e	b	c
TGT = 9 CD = 15	TGT = 8 CD =16	TGT =10 CD = 16	TGT = 14 CD = 20	TGT = 12 CD = 18	TGT = 17 CD = 23

bậc 4
n-i=1

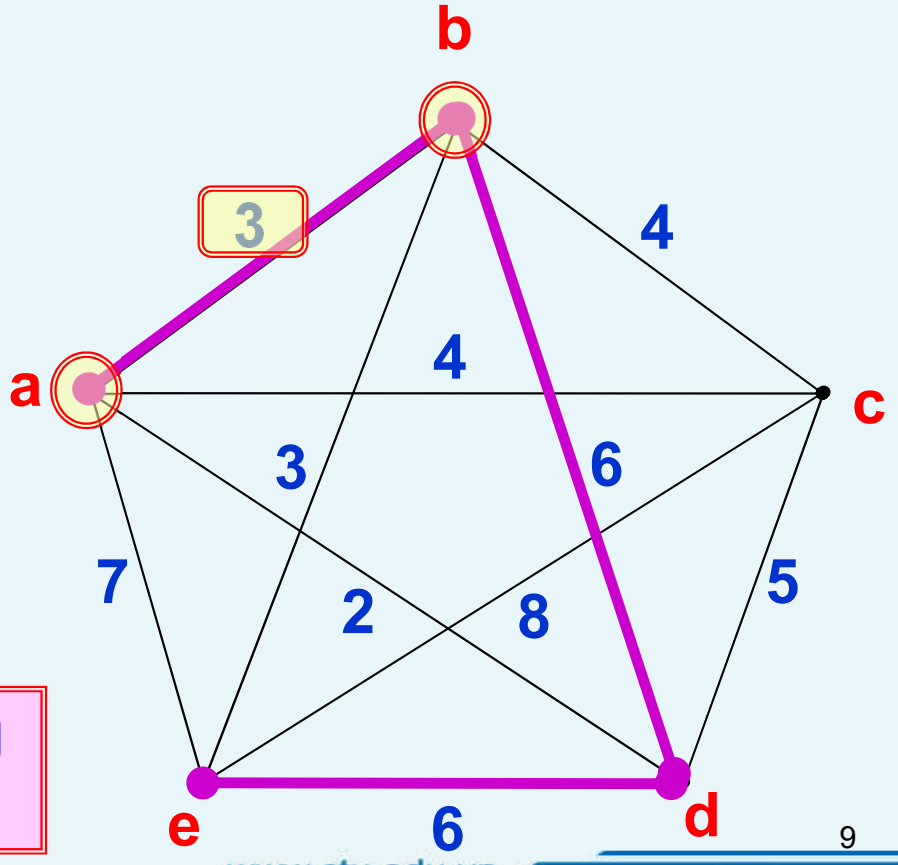
e	c	e
CD = 20 dabced TGT = 23	CD = 20 dabecd TGT = 21	CD = 16 dacbed TGT = 19

GNNTT= 23 → 21 → 19

KHAI BÁO CẠNH

```
typedef struct {
        float do_dai;
        int dau, cuoi;
        int da_dung;
} canh;
```

0: chưa dùng
1: đã dùng



HÀM ĐẶT LẠI CẠNH

```
void reset (canh a[ ][size], int n){
    int i, j;
    for (i = 0; i < n; i++)
        for (j = 0; j < n; j++)
            a[i][j].da_dung = 0;
}
```

Hàm đặt lại tất cả các cạnh trong ma trận cạnh a là chưa dùng

HÀM XÁC ĐỊNH CẠNH NHỎ NHẤT

```
float canh_NN (canh a[ ][size], int n){
  float Cmin = 3.40282e + 38;
  int i, j;
  for (i = 0; i < n; i ++)
    for (j = 0; j < n; j ++)
      if (i != j && !a[i][j].da_dung && a[i][j].do_dai < Cmin)
                Cmin = a[i][j].do_dai;
   return Cmin; }
```

Hàm trả về độ dài *cạnh nhỏ nhất* trong các cạnh chưa dùng

Số thực ∞ → giá trị số thực dấu chấm động lớn nhất 32 bits

HÀM TÍNH CẬN DƯỚI

- Nút trong (**bậc i**):

CD = TGT + (n – i) * Độ dài cạnh nhỏ nhất kế tiếp

(trong số các cạnh chưa sử dụng)

i : bậc của nút

```
float can_duoi (canh a[ ][size], float TGT, int n, int i){
        return TGT+ (n - i) * canh_NN(a, n);
}
```

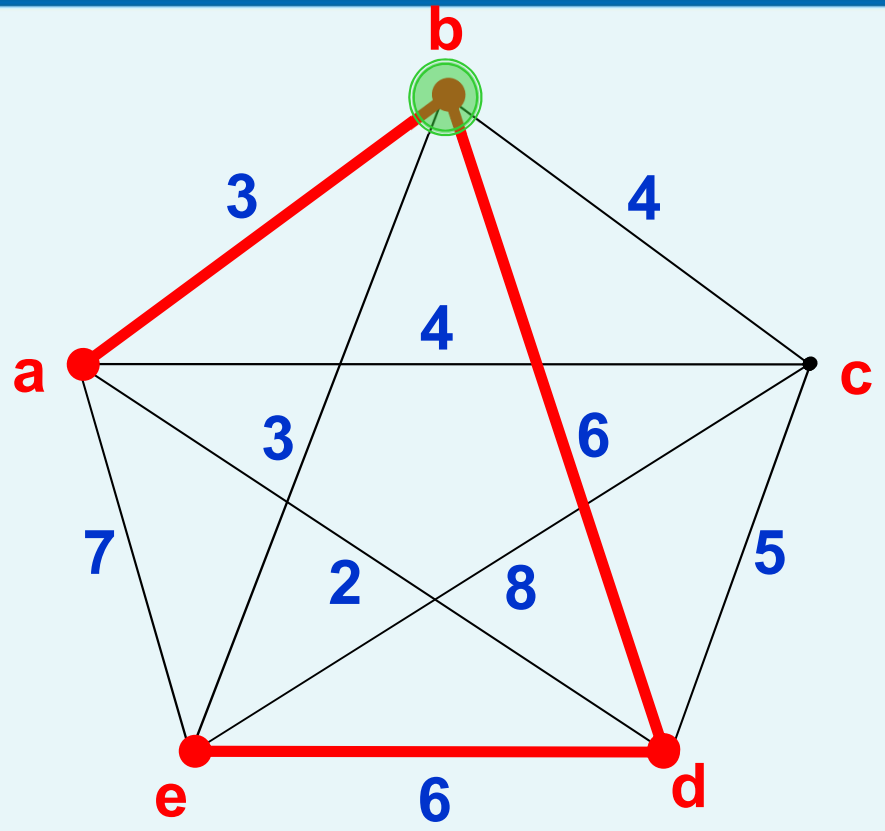
n – i : số cạnh còn lại trong PA

VẤN ĐỀ TẠO CHU TRÌNH

- Giả sử ta đã có một phương án thành phần PA, với k cạnh. Nếu đi tiếp đến đỉnh **ke_tiep** thì có tạo thành chu trình không ?
- Vì các cạnh trong PA đã "gối đầu" lên nhau (***cuối*** cạnh trước ≅ ***đầu*** cạnh sau) nên PA sẽ có chu trình ⇔ đỉnh **ke_tiep** **trùng với *đỉnh đầu*** của một cạnh nào đó trong PA

VÍ DỤ TẠO CHU TRÌNH

- Giả sử PA đã có 3 cạnh **ab, bd** và **de**, bây giờ nếu đi tiếp đến đỉnh **b** sẽ tạo thành chu trình thiếu **b → d → e → b**
- Sở dĩ như vậy là vì đỉnh ke_tiep **b** trùng với đỉnh đầu của cạnh **bd**



HÀM XÁC ĐỊNH CÓ CHU TRÌNH

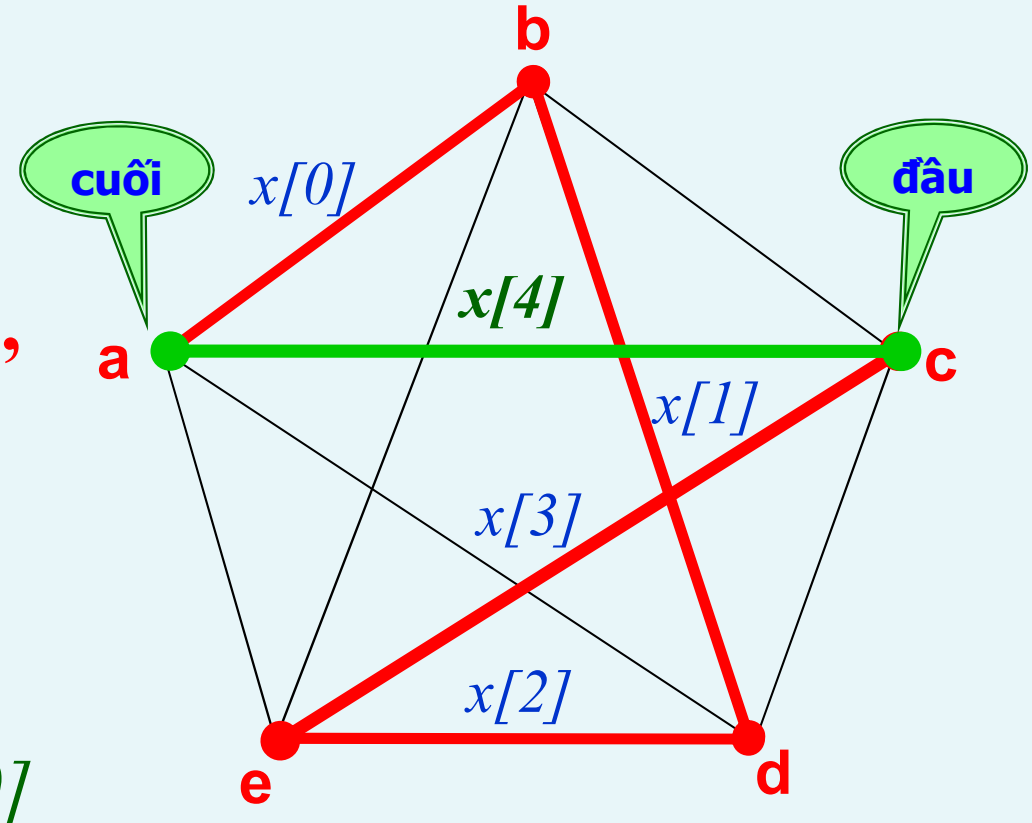
```
int co_chu_trinh (canh x[ ], int k, int ke_tiep){
        int i = 0, co_CT = 0;
        while (i < k && ! co_CT)
                if (ke_tiep = = x[i].dau) co_CT = 1;
                else i++;
        return co_CT;
}
```

CẬP NHẬT PHƯƠNG ÁN

- n = 5 → Chu trình có 5 cạnh từ cạnh x[0] → x[4] *(x[n-1])*
- Nếu tập cạnh x đã có 4 cạnh từ x[0] → x[3] *(x[n-2])* là **ab, bd, de, ec** ⇒ giờ chỉ thêm cạnh cuối **x[4]** *(x[n-1])* với:

- đầu = *cuối của cạnh cuối x[3] (x[n-2])*

- cuối = *đầu của cạnh đầu x[0]*



CANTHO UNIVERSITY

HÀM CẬP NHẬT PHƯƠNG ÁN

```
void Cap_Nhat_PA_TNTT(canh a[ ][size], int n,
float TGT, float *GNNTT, canh x[ ], canh PA[ ]) {
    int i;
     x[n-1] = a[x[n-2].cuoi][x[0].dau];
     TGT= TGT + x[n-1].do_dai;
     if (*GNNTT > TGT) {
          *GNNTT = TGT;
          for (i = 0; i<n; i++) PA[i]=x[i]; } }
```

đầu

cuối