CANTHO UNIVERSITY

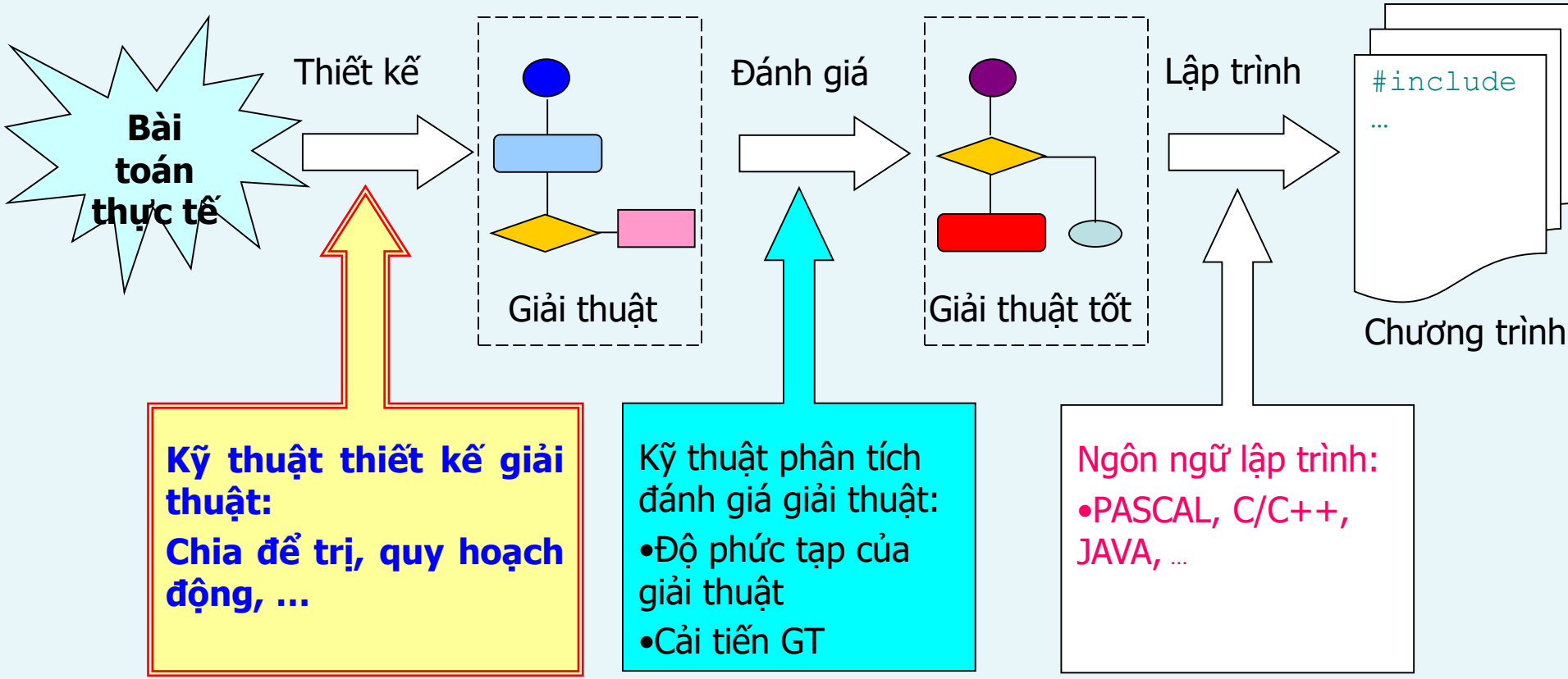
CHƯƠNG 3
KỸ THUẬT THIẾT KẾ THUẬT TOÁN

Bộ môn CÔNG NGHỆ PHẦN MỀM
Khoa Công nghệ thông tin và Truyền thông
Đại học Cần Thơ

Mục tiêu

- Biết các kỹ thuật thiết kế giải thuật: từ ý tưởng cho đến giải thuật chi tiết.
- Hiểu rõ nguyên lý của các kỹ thuật phân tích thiết kế giải thuật.
- Vận dụng kỹ thuật phân tích thiết kế để giải các bài toán thực tế: các bài toán dạng nào thì có thể áp dụng được kỹ thuật này.

Mô hình từ bài toán đến chương trình



Kỹ thuật chia để trị

- Cần phải giải bài toán có kích thước n.
- Ta chia bài toán ban đầu thành một số bài toán con đồng dạng với bài toán ban đầu có kích thước nhỏ hơn n.
- Giải các bài toán con và tổng hợp lời giải của chúng, ta có được lời giải của bài toán ban đầu.
- Đối với từng bài toán con, ta cũng sẽ áp dụng kỹ thuật này để chia chúng thành các bài toán con nhỏ hơn nữa.
- Quá trình phân chia này sẽ cho chúng ta các bài toán cơ sở.

Nhìn lại giải thuật MergeSort và QuickSort

- **MergeSort:**
  - Phân chia: chia danh sách có n phần tử thành 2 danh sách có n/2 phần tử.
  - Quá trình phân chia sẽ dẫn đến các danh sách chỉ có 1 phần tử, là bài toán cơ sở.
  - Tổng hợp: trộn (merge) 2 danh sách có thứ tự thành một danh sách có thứ tự.
- **QuickSort:**
  - Phân hoạch danh sách ban đầu thành 2 danh sách "bên trái" và "bên phải".
  - Sắp xếp 2 danh sách "bên trái" và "bên phải" ta thu được danh sách có thứ tự.
  - Bài toán cơ sở: Sắp xếp danh sách có 1 phần tử hoặc nhiều phần tử có giá trị giống nhau.
  - Tổng hợp: đã bao hàm trong giai đoạn phân chia.

Bài toán nhân số nguyên lớn

- Các NNLT đều có kiểu dữ liệu số nguyên (integer trong Pascal, Int trong C…), nhưng các kiểu này đều có miền giá trị hạn chế.
- Người lập trình phải tìm một cấu trúc dữ liệu thích hợp để biểu diễn cho một số nguyên.
- Để thao tác được trên các số nguyên được biểu diễn bởi một cấu trúc mới, người lập trình phải xây dựng các phép toán cho số nguyên như phép cộng, phép trừ, phép nhân…
- Sau đây ta sẽ đề cập đến bài toán nhân hai số nguyên lớn..

Giải thuật nhân 2 số nguyên lớn

- Xét bài toán nhân 2 số nguyên lớn **X** và **Y**, mỗi số có **n** chữ số.
- Theo cách nhân thông thường:

```
      1426
       x
       3219
     ----------
     12834
     1426
     2852
     4278
   ------------
    4590294
```

- Việc nhân từng chữ số của X và Y tốn n^2 phép nhân.
- Nếu phép nhân 2 chữ số tốn O(1) thời gian thì độ phức tạp của giải thuật này là $O(n^2)$.

Giải thuật chia để trị cho bài toán nhân số nguyên lớn

- Để đơn giản cho việc phân tích giải thuật ta giả sử **n là lũy thừa của 2**.
- Còn về phương diện lập trình, giải thuật cũng đúng trong trường hợp **n bất kỳ**.
- Biểu diễn $X = A10^{n/2} + B$ và $Y = C10^{n/2} + D$
- Trong đó A, B, C, D là các số có n/2 chữ số.
- Ví dụ X = 1234 thì A = 12 và B = 34 vì $12*10^2 + 34 = 1234 = X$
- Với cách biểu diễn này thì $XY = AC10^n + (AD + BC)10^{n/2} + BD$

Giải thuật chia để trị cho bài toán nhân số nguyên lớn (tt)

- Ta phân tích bài toán ban đầu thành một số bài toán nhân 2 số có n/2 chữ số.
- Sau đó, ta kết hợp các kết quả trung gian theo công thức $XY = AC10^{n} + (AD + BC)10^{n/2} + BD$.
- Việc phân chia này sẽ dẫn đến các bài toán nhân 2 số có 1 chữ số. Đây là bài toán cơ sở.

Đánh giá giải thuật

- Theo công thức $XY = AC10^n + (AD + BC)10^{n/2} + BD$
- Ta thực hiện 4 phép nhân các số nguyên có n/2 chữ số, 3 phép cộng các số lớn hơn n chữ số và 2 phép nhân với 10^n và $10^{n/2}$.
- Phép cộng các số có lớn hơn n chữ số cần O(n).
- Phép nhân với 10^n tốn O(n) (dịch trái n lần).
- Gọi T(n) là thời gian nhân 2 số có n chữ số ta có phương trình đệ quy sau:
- $T(1) = 1$
- $T(n) = 4T(n/2) + n$
- Giải hệ này ta được $T(n) = O(n^2)$. Ta không cải tiến được so với giải thuật nhân thông thường.

Cải tiến giải thuật

- Ta biến đổi công thức $XY = AC10^n + (AD + BC)10^{n/2} + BD$
- $XY = AC10^n + [(A - B)(D - C) + AC + BD]10^{n/2} + BD$ (**)
- Theo công thức này, ta chỉ cần 3 phép nhân các số nguyên có n/2 chữ số, 6 phép cộng trừ và 2 phép nhân với 10^n, $10^{n/2}$. Ta có phương trình đệ quy sau:
- $T(1) = 1$
- $T(n) = 3T(n/2) + n$
- Giải phương trình ta được $T(n) = O(n^{\log 3}) = O(n^{1.59})$. Rõ ràng cải tiến hơn giải thuật cũ rất nhiều.

Giải thuật thô để nhân 2 số nguyên có n chữ số

```
Big_Integer mult(Big_Integer X, Big_Integer Y, int n) {
    Big_Integer m1, m2, m3, A, B, C, D;
    int s; /* lưu dấu của tích XY */
    s = sign(X)*sign(Y); /* sign(X) trả về 1 nếu X dương; -1 nếu X âm; 0 nếu X = 0*/
    X = ABS(X);
    Y = ABS(Y);
    if (n == 1) return X*Y*s;
    A = left(X, n/2);
    B = right(X, n/2);
    C = left(Y, n/2);
    D = right(Y, n/2);
    m1 = mult(A, C, n/2);
    m2 = mult(A – B, D – C, n/2);
    m3 = mult(B, D, n/2);
    return s*(m1*10^n + (m1 + m2 + m3)*10^(n/2) + m3);
}
```

Bài toán xếp lịch thi đấu thể thao

- Bài toán đặt ra là xếp lịch thi đấu vòng tròn 1 lượt cho n đấu thủ. Mỗi đấu thủ phải đấu với n-1 đấu thủ còn lại và mỗi đấu thủ chỉ đấu nhiều nhất là 1 trận mỗi ngày. Yêu cầu xếp lịch sao cho số ngày thi đấu là ít nhất.
- Tổng số trận đấu là n(n-1)/2.
- Nếu n chẵn, ta có thể xếp n/2 cặp đấu với nhau mỗi ngày và số ngày thi đấu ít nhất sẽ là n-1 ngày. Ngược lại nếu n lẻ, thì n-1 chẵn, ta có thể xếp (n-1)/2 trận mỗi ngày và vì vậy chúng ta cần n ngày.
- Giả sử $\mathbf{n = 2^k}$ thì n chẵn do đó ta cần ít nhất n - 1 ngày.

Giải thuật chia để trị cho bài toán xếp lịch thi đấu

- Lịch thi đấu là 1 bảng gồm n dòng (tương ứng với n đấu thủ) và n-1 cột (tương ứng với n-1 ngày). Ô (i,j) biểu diễn đấu thủ mà i phải đấu trong ngày j.
- Chia để trị: thay vì xếp cho n người, ta sẽ xếp cho n/2 người sau đó dựa trên kết của lịch thi đấu của n/2 người ta xếp cho n người.
- Quá trình phân chia sẽ dừng lại khi ta phải xếp lịch cho 2 đấu thủ. Việc xếp lịch cho 2 đấu thủ rất dễ dàng: ta cho 2 đấu thủ này thi đấu 1 trận trong 1 ngày.
- Bước khó khăn nhất chính là bước xây dựng lịch cho 4, 8, 16, ... đấu thủ từ lịch thi đấu của 2 đấu thủ.

Xây dựng lịch thi đấu

2 đấu thủ

	1
1	2
2	1

4 đấu thủ

	1	2	3
1	2	3	4
2	1	4	3
3	4	1	2
4	3	2	1

8 đấu thủ

	1	2	3	4	5	6	7
1	2	3	4	5	6	7	8
2	1	4	3	6	5	8	7
3	4	1	2	7	8	5	6
4	3	2	1	8	7	6	5
5	6	7	8	1	2	3	4
6	5	8	7	2	1	4	3
7	8	5	6	3	4	1	2
8	7	6	5	4	3	2	1

Bài toán con cân bằng

- Sẽ tốt hơn nếu ta chia bài toán cần giải thành các bài toán con có kích thước gần bằng nhau.
- Ví dụ: MergeSort phân chia bài toán thành hai bài toán con có cùng kích thước n/2 và do đó thời gian của nó chỉ là O(nlogn). Ngược lại trong trường hợp xấu nhất của QuickSort, khi mảng bị phân hoạch lệch thì thời gian thực hiện là $O(n^2)$.
- Nguyên tắc chung: Chia bài toán thành các bài toán con có kích thước xấp xỉ bằng nhau thì hiệu suất sẽ cao hơn.

Kỹ thuật "tham ăn" (greedy)

- Đây là một kỹ thuật được dùng nhiều để giải các bài toán tối ưu tổ hợp.
- Áp dụng kỹ thuật này tuy không cho chúng ta lời giải tối ưu nhưng sẽ cho một lời giải "tốt"; bù lại chúng ta được lợi khá nhiều về thời gian.

Bài toán tối ưu tổ hợp

- Cho hàm f(X) xác định trên một tập hữu hạn các phần tử D. Hàm f(X) được gọi là hàm mục tiêu.
- Mỗi phần tử $X \in D$ có dạng X = (x1, x2, .. xn) được gọi là một phương án.
- Cần tìm một phương án $X^* \in D$ sao cho f(X*) đạt min (max). Phương án X* như thế được gọi là phương án tối ưu.
- Phương pháp "vét cạn" cần một thời gian mũ.

Nội dung kỹ thuật tham ăn

- Kỹ thuật tham ăn thường được vận dụng để giải bài toán tối ưu tổ hợp bằng cách xây dựng một phương án X.
- Phương án X được xây dựng bằng cách lựa chọn từng thành phần Xi của X cho đến khi hoàn chỉnh (đủ n thành phần).
- Với mỗi Xi, ta sẽ chọn Xi tối ưu. Với cách này thì có thể ở bước cuối cùng ta không còn gì để chọn mà phải chấp nhận một giá trị cuối cùng còn lại.
- Áp dụng kỹ thuật tham ăn sẽ cho một giải thuật thời gian đa thức, tuy nhiên nói chung chúng ta chỉ đạt được **một phương án tốt chứ chưa hẳn là tối ưu**.

Bài toán trả tiền của máy rút tiền tự động ATM

- Trong máy ATM, có sẵn các loại tiền có mệnh giá 100.000 đồng, 50.000 đồng, 20.000 đồng và 10.000 đồng.
- Giả sử mỗi loại tiền đều có số lượng không hạn chế.
- Khi có một khách hàng cần rút một số tiền **n** đồng (tính chẵn đến 10.000 đồng, tức là **n** chia hết cho 10.000).
- Hãy tìm một phương án trả tiền sao cho trả đủ **n** đồng và số tờ giấy bạc phải trả là ít nhất.

Kỹ thuật Tham ăn giải bài toán trả tiền của máy ATM

- Gọi X = (X1, X2, X3, X4) là một phương án trả tiền.
- X1 là số tờ giấy bạc 100.000 đồng, X2 là số tờ giấy bạc 50.000 đồng, X3 là số tờ giấy bạc 20.000 đồng và X4 là số tờ giấy bạc 10.000 đồng.
- Theo yêu cầu ta phải có X1 + X2 + X3 + X4 nhỏ nhất
- X1*100.000+X2*50.000+X3*20.000+X4*10.000 = n.

Kỹ thuật Tham ăn giải bài toán trả tiền của máy ATM (tt)

- Để (X1 + X2 + X3 + X4) nhỏ nhất thì các tờ giấy bạc mệnh giá lớn phải được chọn nhiều nhất.
- Trước hết ta chọn tối đa các tờ giấy bạc 100.000 đồng, nghĩa là X1 là số nguyên lớn nhất sao cho X1 * 100.000 ≤ **n**. Tức là X1 = **n** DIV 100.000.
- Xác định số tiền cần rút còn lại là hiệu n – X1 * 100000
- Chuyển sang chọn loại giấy bạc 50.000 đồng, và cứ tiếp tục như thế cho các loại mệnh giá khác

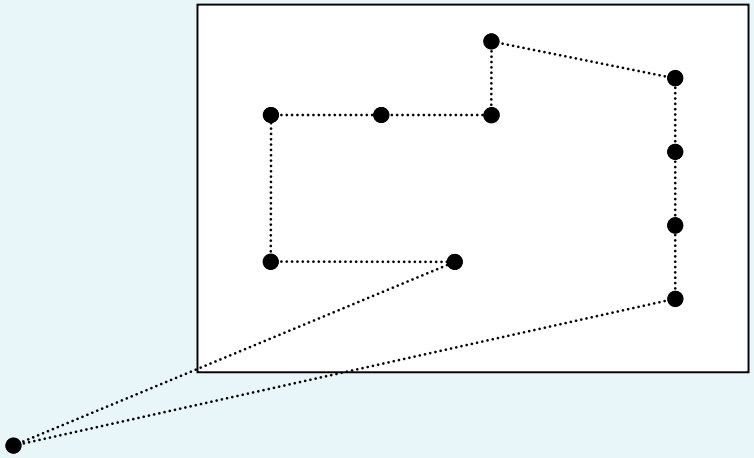
Bài toán trả tiền của máy rút tiền tự động ATM: Ví dụ

- n = 1290000, phương án trả tiền như sau:
- X1 = 1290000 /100000 = 12
- Số tiền còn lại là 1290000 – 12 * 100000 = 90000
- X2 = 90000 / 50000 = 1
- Số tiền còn lại là 90000 – 1 * 50000 = 40000
- X3 = 40000 / 20000 = 2
- Số tiền còn lại là 40000 – 2 * 20000 = 0
- X4 = 0 / 10000 = 0
- Ta có X = (12, 1, 2, 0)

Bài toán đường đi của người giao hàng (TSP): Mô tả bài toán

- Có một người giao hàng cần đi giao hàng tại n thành phố.
- Xuất phát từ một thành phố nào đó, đi qua các thành phố khác để giao hàng và trở về thành phố ban đầu.
- Mỗi thành phố chỉ đến một lần, khoảng cách từ một thành phố đến các thành phố khác là xác định được.
- Giả thiết rằng mỗi thành phố đều có đường đi đến các thành phố còn lại.
- Khoảng cách giữa hai thành phố có thể là khoảng cách địa lý, có thể là cước phí di chuyển hoặc thời gian di chuyển. Ta gọi chung là độ dài.
- Hãy tìm một chu trình sao cho tổng độ dài các cạnh là nhỏ nhất. Hay còn nói là tìm một phương án có giá nhỏ nhất

TSP: Một ứng dụng



Tấm kim loại

Vị trí hàn

Bài toán hàn các điểm trên một tấm kim loại

TSP: Phương pháp vét cạn

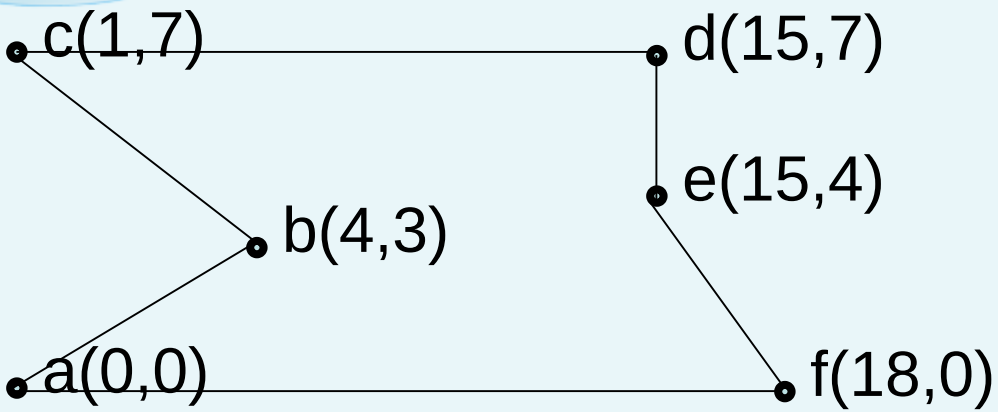
- Ta xét tất cả các chu trình, mỗi chu trình tính tổng độ dài các cạnh của nó rồi chọn một chu trình có tổng độ dài nhỏ nhất.
- Tuy nhiên chúng ta cần xét tất cả là (n-1)!/2 chu trình.
- Đó là một giải thuật thời gian mũ !

TSP: Kỹ thuật tham ăn

1. Sắp xếp các cạnh theo thứ tự tăng của độ dài.
2. Xét các cạnh có độ dài từ nhỏ đến lớn để đưa vào chu trình.
3. Một cạnh sẽ được đưa vào chu trình nếu:
   - Không tạo thành một chu trình thiếu
   - Không tạo thành một đỉnh có cấp ≥ 3
4. Lặp lại bước 3 cho đến khi xây dựng được một chu trình.

- Với kỹ thuật này ta chỉ cần **n(n-1)/2** phép chọn nên ta có một giải thuật cần **$O(n^2)$** thời gian.

TSP: Ví dụ



TT	Canh	X1	Y1	X2	Y2	Do dai
1	de	15	7	15	4	3.00
2	ab	0	0	4	3	5.00
3	bc	4	3	1	7	5.00
4	ef	15	4	18	0	5.00
5	ac	0	0	1	7	7.07
6	df	15	7	18	0	7.62
7	be	4	3	15	4	11.05
8	bd	4	3	15	7	11.70
9	cd	1	7	15	7	14.00
10	bf	4	3	18	0	14.32
11	ce	1	7	15	4	14.32
12	ae	0	0	15	4	15.52
13	ad	0	0	15	7	16.55
14	af	0	0	18	0	18.00
15	cf	1	7	18	0	18.38

TSP: Giải thuật

```
void TSP() {
    /*E là tập hợp các cạnh, Chu_trinh là tập hợp các cạnh được chọn
    để đưa vào chu trình, mở đầu Chu_trinh rỗng*/
    /*Sắp xếp các cạnh trong E theo thứ tự tăng của độ dài*/
    Chu_Trinh = Φ;
    Gia = 0.0;
    while (E != Φ) {
                    if (cạnh e có thể chọn) {
                    Chu_Trinh = Chu_Trinh + [e];
                    Gia = Gia + độ dài của e;
          }
                    E := E-[e];
    }
}
```

TSP: Một cách tiếp cận khác

1. Xuất phát từ một đỉnh bất kỳ, chọn một cạnh có độ dài nhỏ nhất trong tất cả các cạnh đi ra từ đỉnh đó để đến đỉnh kế tiếp.
2. Từ đỉnh kế tiếp ta lại chọn một cạnh có độ dài nhỏ nhất đi ra từ đỉnh này thoả mãn hai điều kiện nói trên để đi đến dỉnh kế tiếp.
3. Lặp lại bước 2 cho đến khi đi tới đỉnh **n** thì quay trở về đỉnh xuất phát.

Bài toán cái ba lô

Cho một cái ba lô có thể đựng một trọng lượng **W** và **n** loại đồ vật, mỗi đồ vật **i** có một trọng lượng **gi** và một giá trị **vi**. Tất cả các loại đồ vật đều có số lượng không hạn chế. Tìm một cách lựa chọn các đồ vật đựng vào ba lô, chọn các loại đồ vật nào, mỗi loại lấy bao nhiêu sao cho tổng trọng lượng không vượt quá **W** và tổng giá trị là lớn nhất.

Mô hình hóa

- Gọi $X=(X_1, X_2, \ldots X_n)$ với X_i là số nguyên không âm, là một phương án. Xi là số đồ vật thứ i.
- Cần tìm X sao cho:
  - $X_1g_1 + X_2g_2 + \ldots + X_ng_n <= W$
  - $F(X) = X_1v_1 + X_2v_2 + \ldots + X_nv_n \rightarrow Max$

Bài toán cái ba lô: GT tham ăn

- Tính đơn giá cho các loại đồ vật.
- Xét các loại đồ vật theo **thứ tự đơn giá từ lớn đến nhỏ**.
- Với mỗi đồ vật được xét sẽ lấy một số lượng tối đa mà trọng lượng còn lại của ba lô cho phép.
- Xác định trọng lượng còn lại của ba lô và quay lại bước 3 cho đến khi không còn có thể chọn được đồ vật nào nữa.

Bài toán cái ba lô: ví dụ

Ta có một ba lô có trọng lượng là 37 và 4 loại đồ vật với trọng lượng và giá trị tương ứng được cho trong bảng bên dưới:

Loại đồ vật	Trọng lượng	Giá trị
A	15	30
B	10	25
C	2	2
D	4	6

Bài toán cái ba lô: Vét cạn

Loại đồ vật	Trọng lượng	Giá trị
A	15	30
B	10	25
C	2	2
D	4	6

Bài toán cái ba lô: ví dụ

ĐV	TL	GT	ĐG
B	10	25	2.5
A	15	30	2.0
D	4	6	1.5
C	2	2	1.0

- Phương án là X=(Xa,Xb,Xc,Xd)
- Xb = 37/10 = 3
- W= 37 - 3*10 = 7.
- Xa = 7/15 =0
- Xd = 7/4 = 1
- W = 7-4 = 3.
- Xc = 3/2 = 1.
- W = 3 – 2 = 1
- TTL là 3*10 + 1*4 + 1*2 = 36
- TGT là 3*25+1*6+1*2 = 83.

BT cái ba lô: tổ chức dữ liệu

- Mỗi đồ vật được biểu diễn bởi một mẩu tin có các trường:
  - Ten: Lưu trữ tên đồ vật.
  - Trong_luong: Lưu trữ trọng lượng của đồ vật.
  - Gia_tri: Lưu trữ giá trị của đồ vật
  - Don_gia: Lưu trữ đơn giá của đồ vật
  - Phuong_an: Lưu trữ số lượng đồ vật được chọn theo phương án.
- Danh sách các đồ vật được biểu diễn bởi một mảng các đồ vật.

BT cái ba lô: chương trình

```
#define MAX_SIZE   100
typedef struct {
    char Ten [20];
    float Trong_luong, Gia_tri, Don_gia;
    int Phuong_an;
} Do_vat;

typdef Do_vat Danh_sach_do_vat[MAX_SIZE];

void Greedy (Danh_sach_do_vat dsdv, float W) {
    int i;
    /*Sắp xếp mảng dsdv theo thứ tự giảm của don_gia*/
    for (i = 0; i < n; i++) {
        dsdv[i].Phuong_an = Chon(dsdv[i].Trong_luong,
    W);
        W = W – dsdv[i].phuong_an * dsdv[i].Trong_luong;
    }
```

Biến thể của bài toán cái ba lô

- Có một số biến thể của bài toán cái ba lô như sau:
  - Mỗi đồ vật **i** chỉ có một số lượng **si**. Với bài toán này khi lựa chọn vật **i** ta không được lấy một số lượng vượt quá **si**.
  - Mỗi đồ vật chỉ có **một cái**. Với bài toán này thì với mỗi đồ vật ta chỉ có thể **chọn hoặc không chọn**.

Quy hoạch động: nội dung kỹ thuật

- Trong giải thuật đệ quy, một số bài toán con nào đó bị giải **nhiều lần**.
- Tạo ra một **bảng để lưu trữ** kết quả của các bài toán con và khi cần chúng ta sẽ sử dụng kết quả đã được lưu trong bảng mà **không cần phải giải lại bài toán đó**.
- Tạo bảng bằng cách:
  - Gán giá trị cho một số ô nào đó.
  - Gán trị cho các ô khác nhờ vào giá trị của các ô trước đó.
- Tra bảng và xác định kết quả của bài toán ban đầu.

Quy hoạch động: ưu và nhược điểm

- Ưu điểm của phương pháp quy hoạch động là:
  - Chương trình thực hiện nhanh.
  - Kỹ thuật quy hoạch động có thể vận dụng để giải các bài toán tối ưu, các bài toán có công thức truy hồi.
- Quy hoạch động sẽ không đem lại hiệu quả khi:
  - Không tìm được công thức truy hồi.
  - Số lượng các bài toán con cần giải quyết và lưu giữ kết quả là rất lớn.
  - Sự kết hợp lời giải của các bài toán con chưa chắc cho ta lời giải của bài toán ban đầu.

Bài toán tính số tổ hợp

- Tính số tổ hợp chập k của n theo công thức truy hồi:

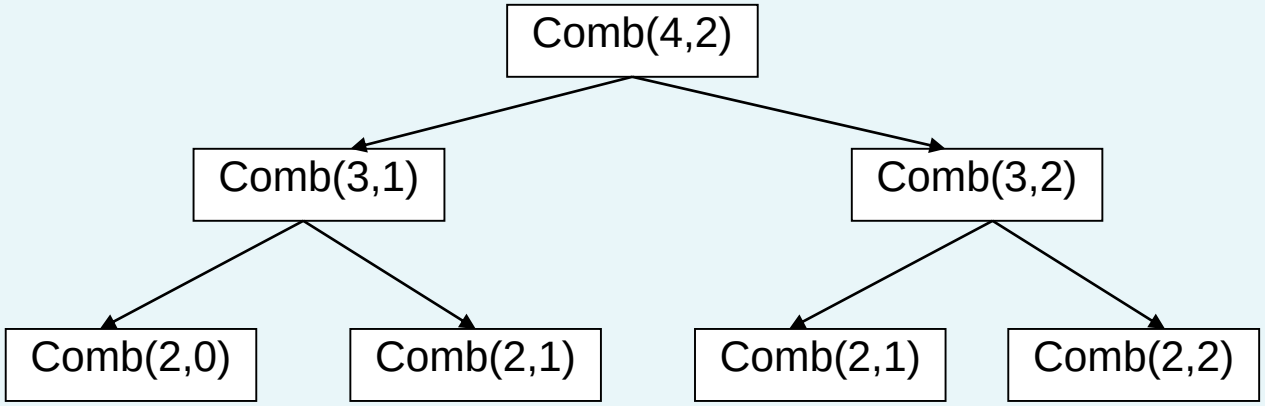
$$C^{k}_{n} = \begin{cases} 1 \text{ neu } k = 0 \text{ hoac } k = n \\ C^{k-1}_{n-1} + C^{k}_{n-1} \end{cases}$$

Bài toán tính số tổ hợp: giải thuật đệ quy

```
int Comb(int n, int k) {
   if (k==0 || k==n)
        return 1;
   return Comb(n-1, k-1) + Comb(n-1,k);
}
```

Bài toán tính số tổ hợp: phân tích giải thuật

- Gọi T(n) là thời gian để tính số tổ hợp chập k của n, thì ta có phương trình đệ quy:
- T(1) = C1 và T(n) = 2T(n-1) + C2
- Giải phương trình này ta được **T(n) = O(2^n)**



Bài toán tính số tổ hợp: kỹ thuật quy hoạch động

- Xây dựng một bảng C gồm n+1 dòng (từ 0 đến n) và n+1 cột (từ 0 đến n).
- C(i,j) lưu trữ giá trị của Comb(i,j), theo quy tắc sau: (Quy tắc tam giác Pascal):
  - C(0,0) = 1;
  - C(i,0) =1;
  - C(i,i) = 1 với $0 < i \leq n$;
  - C(i,j) = C(i-1,j-1) + C(i-1,j) với $0 < j < i \leq n$.
- C(n,k) chính là Comb(n,k).

Bài toán tính số tổ hợp: Tam giác Pascal tính Comb(4,2)

i \ j	0	1	2	3	4
0	1				
1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	4	6	4	1

Bài toán tính số tổ hợp: Kỹ thuật quy hoạch động

```
int Comb(int n, int k) {
        int C[n+1,n+1], i, j;
/*1*/     C[0,0] = 1;
/*2*/     for (i = 1; i <= n; i++) {
/*3*/       C[i,0] = 1;
/*4*/       C[i,i] = 1;
/*5*/       for (j = 1; j < i; j++)
            C[i,j]=C[i-1,j-1]+C[i-1,j];
        }
/*6*/     return C[n,k];
    }
```

i \ j	0	1	2	3	4
0					
1					
2					
3					
4					

Bài toán tính số tổ hợp: nhận xét

- Thông qua việc xác định độ phức tạp, ta thấy rõ ràng giải thuật quy hoạch động hiệu quả hơn nhiều so với giải thuật đệ qui ($n^2 < 2^n$).
- Tuy nhiên việc sử dụng bảng (mảng hai chiều) như trên còn lãng phí ô nhớ, do đó ta sẽ cải tiến thêm một bước bằng cách sử dụng véctơ (mảng một chiều) để lưu trữ kết quả trung gian.

i \ j	0	1	2	3	4
0					
1					
2					
3					
4					

Giải thuật tiết kiệm bộ nhớ

- Ta sẽ dùng một véctơ V có n+1 phần tử từ V[0] đến V[n].
- Trong đó, ở bước I, V[j] lưu trữ giá trị C^j_i (j = 0 đến i).
- V chỉ lưu trữ được một dòng và ở bước cuối cùng, V lưu trữ các giá trị ứng với i = n, trong đó V[k] chính là C^k_n.
- Khởi đầu, ứng với i =1, ta cho V[0] = 1 và V[1] = 1. Tức là $C^0_1 = 1$ và $C^1_1 = 1$.
- Với các giá trị i từ 2 đến n, ta thực hiện như sau:
  - V[0] được gán giá trị 1 tức là $C^0i = 1$. Tuy nhiên giá trị V[0] = 1 đã được gán ở trên.
  - Với j từ 1 đến i-1, ta vẫn áp dụng công thức $C^j_i = C^{j-1}_{i-1} + C^j_{i-1}$. Tuy nhiên do chỉ có một véctơ V và lúc này nó sẽ lưu trữ các giá trị của dòng i, tức là dòng i-1 sẽ không còn. Để khắc phục điều này ta dùng thêm hai biến trung gian p1 và p2. p1 lưu trữ C^{j-1}i-1 và p2 lưu trữ C^j_{i-1}. Khởi đầu p1 được gán V[0] tức là C^0i-1 và p2 được gán V[j] tức là C^j_{i-1}, V[j] lưu trữ giá trị C^j_i sẽ được gán bởi p1+p2, sau đó p1 được gán bởi p2, nghĩa là khi j tăng lên 1 đơn vị thành j+1 thì p1 là C^j_{i-1} và nó được dùng để tính C^{j+1}_i.
  - Cuối cùng với j = i ta gán V[i] giá trị 1 tức là C^ii = 1.

Chương trình tiết kiệm bộ nhớ

```
int Comb(int n, int k) {
     int V[n+1];
     int i, j, p1, p2;
/*1*/       V[0] = 1;
/*2*/       V[1] = 1;
/*3*/       for (i = 2; i <= n; i++) {
/*4*/                 p1 = V[0];
/*5*/                 for (j = 1; j < I; j++) {
/*6*/                           p2 = V[j];
/*7*/                           V[j]= p1+p2;
/*8*/                           p1= p2; }
/*9*/                 V[i] = 1;  }
/*10*/      return V[k];
       }
```

Quy hoạch động: bài toán cái ba lô

- Cho một cái ba lô có thể đựng một trọng lượng **W** và **n** loại đồ vật, mỗi đồ vật **i** có một trọng lượng **gi** và một giá trị **vi**. Tất cả các loại đồ vật đều có số lượng không hạn chế. Tìm một cách lựa chọn các đồ vật đựng vào ba lô, chọn các loại đồ vật nào, mỗi loại lấy bao nhiêu sao cho tổng trọng lượng không vượt quá **W** và tổng giá trị là lớn nhất.
- Sử dụng kỹ thuật quy hoạch động để giải bài toán cái ba lô với một lưu ý là các số liệu đều cho dưới dạng **số nguyên**.

Quy hoạch động: bt cái ba lô

- Giả sử X[k,V] là số lượng đồ vật k được chọn, F[k,V] là tổng giá trị của k đồ vật đã được chọn và V là trọng lượng còn lại của ba lô, k = 1..n, V = 0..W.
- Trong trường hợp đơn giản nhất, khi chỉ có một đồ vật, ta tính được X[1,V] và F[1,V] với mọi V từ 0 đến W như sau:
- X[1,V] = V/g1 và F[1,V] = X[1,V] * v1.
- Giả sử ta đã tính được F[k-1,V], khi có thêm đồ vật thứ k, ta sẽ tính được F[k,V], với mọi V từ 0 đến W. Cách tính như sau: Nếu ta chọn xk đồ vật loại k, thì trọng lượng còn lại của ba lô dành cho k-1 đồ vật từ 1 đến k-1 là U = V-xk*gk và tổng giá trị của k loại đồ vật đã được chọn F[k,V] = F[k-1,U] + xk*vk, với xk thay đổi từ 0 đến yk= V/gk và ta sẽ chọn xk sao cho F[k,V] lớn nhất.

Quy hoạch động: bt cái ba lô

- Ta có công thức truy hồi như sau:
- X[1,V] = V/g1 và F[1,V] = X[1,V] * v1.
- F[k,V] = Max(F[k-1,V-xk*gk] + xk*vk) với xk chạy từ 0 đến V/gk.
- Sau khi xác định được F[k,V] thì X[k,V] là xk ứng với giá trị F[k,V] được chọn trong công thức trên.
- Để lưu các giá trị trung gian trong quá trình tính F[k,V] theo công thức truy hồi trên, ta sử dụng một bảng gồm n dòng từ 1 đến n, dòng thứ k ứng với đồ vật loại k và W+1 cột từ 0 đến W, cột thứ V ứng với trọng lượng V. Mỗi cột V bao gồm hai cột nhỏ, cột bên trái lưu F[k,V], cột bên phải lưu X[k,V]. Trong lập trình ta sẽ tổ chức hai bảng tách rời là F và X.

Quy hoạch động: bài toán cái ba lô – ví dụ

Ví dụ bài toán cái ba lô với trọng lượng W=9, và 5 loại đồ vật được cho trong bảng sau

Đồ vật	Trọng lượng (gi)	Giá trị (vi)
1	3	4
2	4	5
3	5	6
4	2	3
5	1	1

Bảng F và X với W=9

Đồ vật	gi	vi
1	3	4
2	4	5
3	5	6
4	2	3
5	1	1

X[1,V] = V DIV g1 và F[1,V] = X[1,V] * v1.

F[k,V] = Max(F[k-1,V-xk*gk] + xk*vk) với xk chạy từ 0 đến V DIV gk.

V / k	0		1		2		3		4		5		6		7		8		9	
1	0	0	0	0	0	0	4	1	4	1	4	1	8	2	8	2	8	2	12	3
2	0	0	0	0	0	0	4	0	5	1	5	1	8	0	9	1	10	2	12	0
3	0	0	0	0	0	0	4	0	5	0	6	1	8	0	9	0	10	0	12	0
4	0	0	0	0	3	1	4	0	6	2	7	1	9	3	10	2	12	4	13	3
5	0	0	1	1	3	0	4	0	6	0	7	0	9	0	10	0	12	0	13	0

Cách tính bảng F và X

V / k	0		1		2		3		4		5		6		7		8		9	
1	0	0	0	0	0	0	4	1	4	1	4	1	8	2	8	2	8	2	12	3
2	0	0	0	0	0	0	4	0	5	1	5	1	8	0	9	1	10	2	12	0
3	0	0	0	0	0	0	4	0	5	0	6	1	8	0	9	0	10	0	12	0
4	0	0	0	0	3	1	4	0	6	2	7	1	9	3	10	2	12	4	13	3
5	0	0	1	1	3	0	4	0	6	0	7	0	9	0	10	0	12	0	13	0

- Điền giá trị cho dòng 1, sử dụng công thức: X[1,V] = V DIV g1 và F[1,V] = X[1,V] * v1.
- Từ dòng 2 đến dòng 5, phải sử dụng công thức truy hồi:
- F[k,V] = Max(F[k-1,V-xk*gk] + xk*vk) với xk chạy từ 0 đến V DIV gk.
- Ví dụ để tính F[2,7], ta có xk chạy từ 0 đến V DIV gk, trong trường hợp này là xk chạy từ 0 đến 7 DIV 4, tức xk có hai giá trị 0 và 1.
- Khi đó F[2,7] = Max (F[2-1, 7-0*4] + 0*5, F[2-1,7-1*4] + 1*5)
- = Max(F[1,7], F[1,3] + 5) = Max(8, 4+5) = 9.
- F[2,7] = 9 ứng với xk = 1 do đó X[2,7] = 1.

Tra bảng để xác định phương án

V / k	0		1		2		3		4		5		6		7		8		9	
1	0	0	0	0	0	0	4	**1**	4	1	4	1	8	2	8	2	8	2	12	3
2	0	0	0	0	0	0	4	**0**	5	1	5	1	8	0	9	1	10	2	12	0
3	0	0	0	0	0	0	4	**0**	5	0	6	1	8	0	9	0	10	0	12	0
4	0	0	0	0	3	1	4	0	6	2	7	1	9	3	10	2	12	4	13	**3**
5	0	0	1	1	3	0	4	0	6	0	7	0	9	0	10	0	12	0	13	**0**

- Khởi đầu, trọng lượng còn lại của ba lô V = W.
- Xét các đồ vật từ n đến 1, với mỗi đồ vật k, ứng với trọng lượng còn lại V của ba lô, nếu X[k,V] > 0 thì chọn X[k,V] đồ vật loại k. Tính lại V = V - X[k,V] * gk.
- Ví dụ, trong bảng trên, ta sẽ xét các đồ vật từ 5 đến 1. Khởi đầu V = W = 9.
- Với k = 5, vì X[5,9] = 0 nên ta không chọn đồ vật loại 5.
- Với k = 4, vì X[4,9] = 3 nên ta chọn 3 đồ vật loại 4. Tính lại V = 9 – 3 * 2 = 3.
- Với k = 3, vì X[3,3] = 0 nên ta không chọn đồ vật loại 3.
- Với k = 2, vì X[2,3] = 0 nên ta không chọn đồ vật loại 2.
- Với k = 1, vì X[1,3] = 1 nên ta chọn 1 đồ vật loại 1. Tính lại V = 3 – 1 * 3 = 0.
- Vậy tổng trọng lương các vật được chọn là 3 * 2 + 1 * 3 = 9.
- Tổng giá trị các vật được chọn là 3 * 3 + 1 * 4 = 13.

GT quy hoạch động cho bài toán cái ba lô: Tổ chức dữ liệu

- Mỗi đồ vật được biểu diễn bởi một mẩu tin có các trường:
  - Ten: Lưu trữ tên đồ vật.
  - Trong_luong: Lưu trữ trọng lượng của đồ vật.
  - Gia_tri: Lưu trữ giá trị của đồ vật
  - Phuong_an: Lưu trữ số lượng đồ vật được chọn theo phương án.
- Danh sách các đồ vật được biểu diễn bởi một mảng các đồ vật.

GT quy hoạch động cho bài toán cái ba lô: Định nghĩa DL

```
#define MAX 10
typedef struct {
    char Ten[20];
    int Trong_luong, Gia_tri;
    int Phuong_an;
} Do_vat;
typedef Do_vat Danh_sach_vat[MAX];
typedef int Bang[MAX][100];
```

Thủ tục tạo bảng

```
void Tao_Bang (Danh_sach_vat ds_vat, int
      n, int W, Bang F, Bang X) {
      int xk, yk, k;
      int FMax, XMax, v;
      /*Lấp đầy hàng đầu tiên của 2 bảng*/
      for (v= 0; v <= W; v++) {
          X[0,v] = v/ds_vat[0].trong_luong;
          F[0,v] = X[0, v] * ds_vat[0].gia_tri;
      }
      /*Lấp đầy các hàng còn lại*/
      for (k = 1; k <= n-1; k++) {
                X[k,0] = 0;
                F[k,0] = 0;
```

```
for ( v=1; v <= W; v++) {
      Fmax = F[k-1,v] ;
      XMax = 0;
      yk = v/ds_vat[k].trong_luong;
      for (xk=1; xk <= yk; xk++)
        if (F[k-1,v-xk*ds_vat[k].trong_luong]+
            xk*ds_vat[k].gia_tri > FMax) {
            FMax=F[k-1,v-
      k*ds_vat[k].trong_luong]+
                        xk*ds_vat[k].gia_tri;
                      XMax= xk;
      }
      F[k,v] = FMax;
      X[k,v] = XMax;
  }
}
}
```

Thủ tục tra bảng

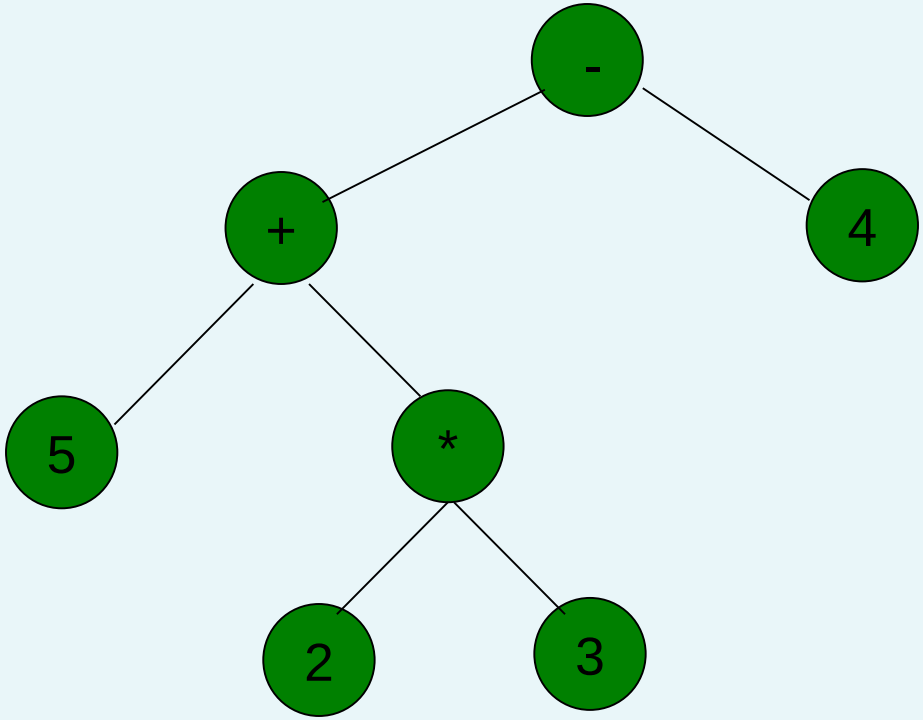
```
void Tra_Bang(Danh_sach_vat ds_vat, int n, int W Bang F,
    Bang X) {
    int k, v;
    v = W;
    for (k = n-1; k >=0; k--)
        if (X[k,v] > 0) {
            ds_vat[k].Phuong_an = X[k,v];
            v = v - X[k, v] * ds_vat[k].trong_luong;
        }
}
```

Kỹ thuật quay lui

- Kỹ thuật quay lui (backtracking) là một quá trình phân tích đi xuống và quay lui trở lại theo con đường đã đi qua.
- Tại mỗi bước phân tích chúng ta chưa giải quyết được vấn đề do còn thiếu cứ liệu nên cứ phải phân tích cho tới các điểm dừng, nơi chúng ta xác định được lời giải của chúng hoặc là xác định được là không thể (hoặc không nên) tiếp tục theo hướng này.
- Từ các điểm dừng này chúng ta quay ngược trở lại theo con đường mà chúng ta đã đi qua để giải quyết các vấn đề còn tồn đọng và cuối cùng ta sẽ giải quyết được vấn đề ban đầu.
- 3 kỹ thuật quay lui: "vét cạn" là kỹ thuật phải đi tới tất cả các điểm dừng rồi mới quay lui. "Cắt tỉa Alpha-Beta" và "Nhánh-Cận" là hai kỹ thuật cho phép chúng ta không cần thiết phải đi tới tất cả các điểm dừng, mà chỉ cần đi đến một số điểm nào đó và dựa vào một số suy luận để có thể quay lui sớm.

Định trị cây biểu thức số học

- Cây biểu thức số học là một cây nhị phân, trong đó các nút lá biểu diễn cho các toán hạng, các nút trong biểu diễn cho các toán tử.
- Biểu thức 5 + 2 * 3 - 4 sẽ được biểu diễn bởi cây trong hình bên



Giải thuật quay lui (vét cạn) cho việc định trị cho cây BTSH

```
float Eval(node n) {
    if (n là lá)
        return (trị của toán hạng trong n);
    return (Toán tử trong n (Eval (Con trái của n), Eval (Con phải của n)));
}
```

- Muốn định trị cho cây biểu thức T, ta gọi Eval(ROOT(T)).

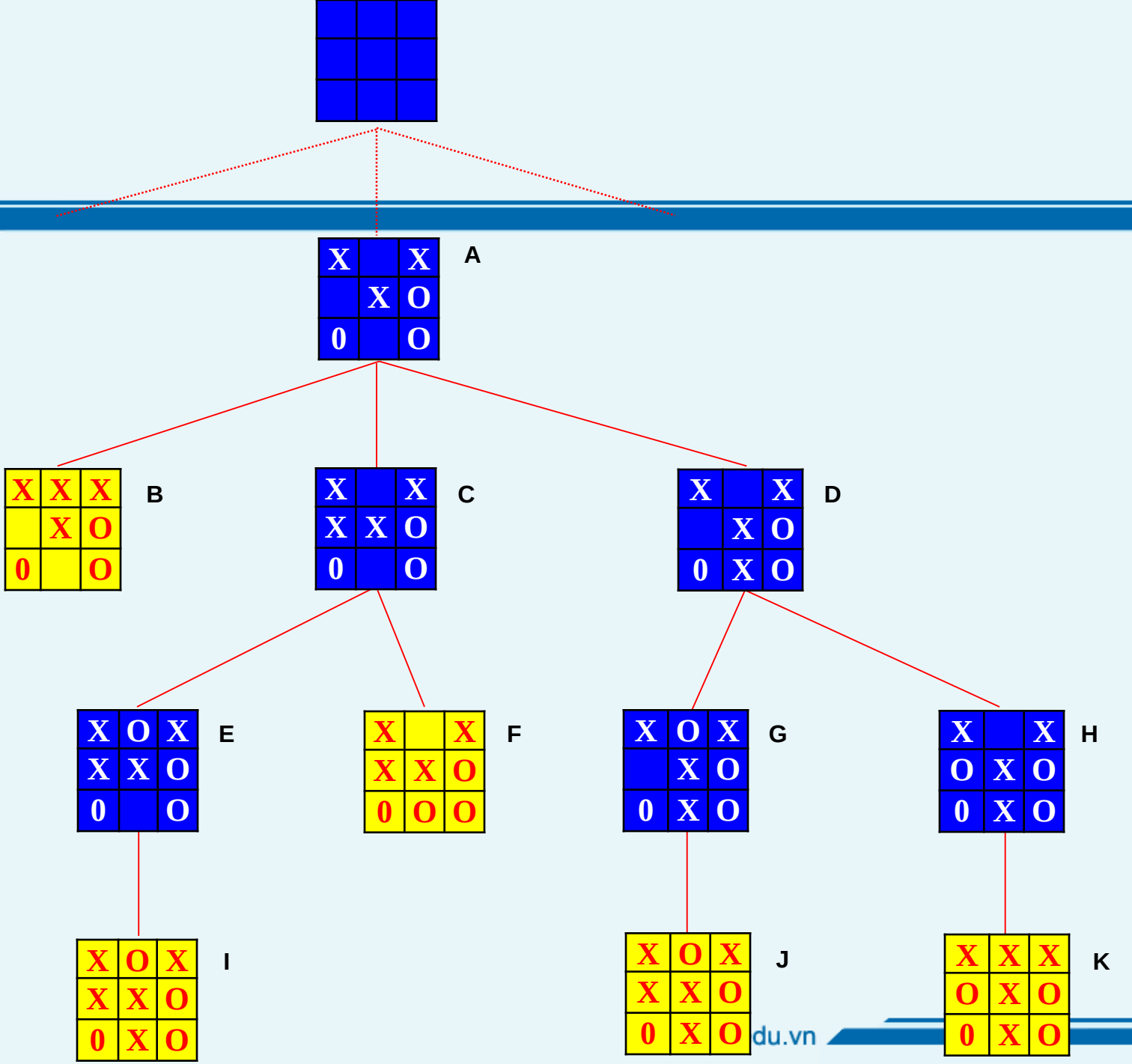
Cây trò chơi: mô tả

- Xét một trò chơi trong đó hai người thay phiên nhau đi nước của mình như cờ vua, cờ tướng, carô...
- Trò chơi có một trạng thái bắt đầu và mỗi nước đi sẽ biến đổi trạng thái hiện hành thành một trạng thái mới.
- Trò chơi sẽ kết thúc theo một quy định nào đó, theo đó thì cuộc chơi sẽ dẫn đến một trạng thái phản ánh có một người thắng cuộc hoặc một trạng thái mà cả hai đấu thủ không thể phát triển được nước đi của mình, ta gọi nó là trạng thái hòa cờ.
- Ta tìm cách phân tích xem từ một trạng thái nào đó sẽ dẫn đến đấu thủ nào sẽ thắng với điều kiện cả hai đấu thủ đều có trình độ như nhau.

Biểu diễn trò chơi bằng cây trò chơi

- Một trò chơi có thể được biểu diễn bởi một cây trò chơi.
- Mỗi một nút của cây biểu diễn cho một trạng thái.
- Nút gốc biểu diễn cho trạng thái bắt đầu của cuộc chơi.
- Mỗi nút lá biểu diễn cho một trạng thái kết thúc của trò chơi (trạng thái thắng thua hoặc hòa).
- Nếu trạng thái x được biểu diễn bởi nút n thì các con của n biểu diễn cho tất cả các trạng thái kết quả của các nước đi có thể xuất phát từ trạng thái x.

X-đi



X-đi

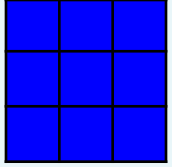
O-đi

X-đi

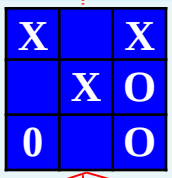
O-đi

CANTHO UNIVERSITY

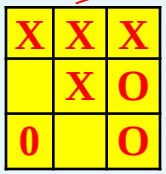
X-đi

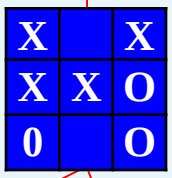


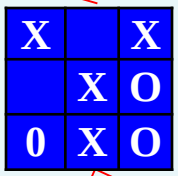
X-đi
Max



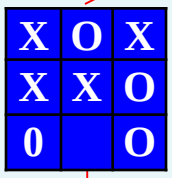
O-đi
Min

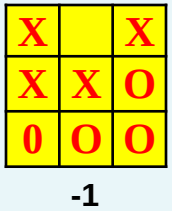


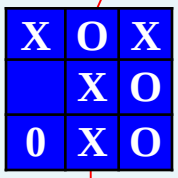


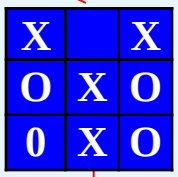


X-đi
Max

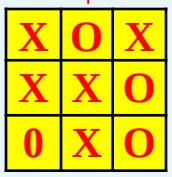


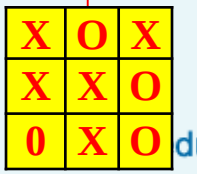


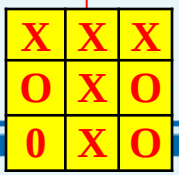




O-đi
Min







Giải thuật vét cạn định trị cây trò chơi: giả thiết

- Ta có một hàm Payoff nhận vào một nút lá và cho ta giá trị của nút lá đó.
- Các hằng ∞ và -∞ tương ứng là các trị Payoff lớn nhất và nhỏ nhất.
- Khai báo kiểu ModeType = (MIN, MAX) để xác định định trị cho nút là MIN hay MAX.
- Một kiểu NodeType được khai báo một cách thích hợp để biểu diễn cho một nút trên cây phản ánh một trạng thái của cuộc chơi.
- Ta có một hàm is_leaf để xác định xem một nút có phải là nút lá hay không?
- Hàm max và min tương ứng lấy giá trị lớn nhất và giá trị nhỏ nhất của hai giá trị.
- Hàm Search nhận vào một nút n và kiểu mode của nút đó (MIN hay MAX) trả về giá trị của nút.

Giải thuật vét cạn định trị cây trò chơi: cài đặt bằng C/C++

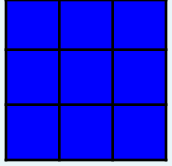
```
float Search(NodeType n, ModeType mode) {
    NodeType C;  /*C là một nút con của nút n*/
    float value;
    /*Lúc đầu ta cho value một giá trị tạm,
    sau khi đã xét hết tất cả các con của nút n thì
    value là giá trị của nút n*/
    if (is_leaf(n))
        return Payoff(n);
    /*Khởi tạo giá trị tạm cho n*/
    if (mode == MAX) value = -∞;
    else value = ∞;
```

```
    /*Xét tất cả các con của n,
    mỗi lần xác định được giá trị của
    một nút con,
    ta phải đặt lại giá trị tạm value.
    Khi đã xét hết tất cả các con thì
    value là giá trị của n*/
    for (với mỗi con C của n)
        if (mode == MAX)
            value = max(value,
    Search(C, MIN));
        else
            value = min(value,
    Search(C, MAX));
    return value;
}
```

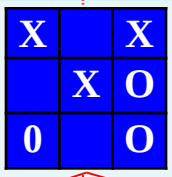
Kỹ thuật cắt tỉa Alpha-Beta (Alpha-Beta Pruning)

- Nếu P là một nút MAX và ta đang xét một nút con Q của nó (dĩ nhiên Q là nút MIN). Nếu $Vp \geq Vq$ cắt các con chưa xét của Q.
- Tương tự nếu P là nút MIN (tất nhiên Q là nút MAX) và $Vp \leq Vq$ thì cắt các con chưa xét của Q.
- Quy tắc định trị cho một nút không phải là nút lá như sau:
  - Khởi đầu nút MAX có giá trị tạm là $-\infty$ và nút MIN có giá trị tạm là ∞.
  - Nếu tất cả các nút con của một nút đã được xét hoặc bị cắt tỉa thì giá trị tạm của nút đó trở thành giá trị của nó.
  - Nếu một nút MAX n có giá trị tạm là V1 và một nút con của nó có giá trị là V2 thì đặt giá trị tạm mới của n là max(V1,V2). Nếu n là nút MIN thì đặt giá trị tạm mới của n là min(V1,V2).
  - Vận dụng quy tắc cắt tỉa Alpha-Beta nói trên để hạn chế số lượng nút phải xét.

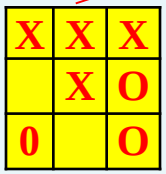
X-đi

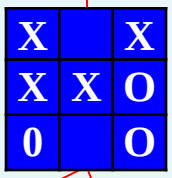


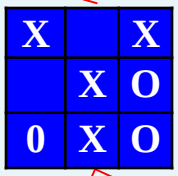
X-đi
Max



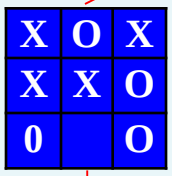
O-đi
Min

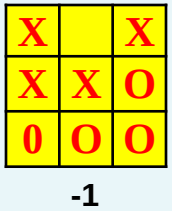


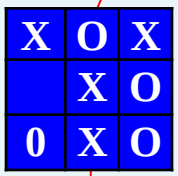


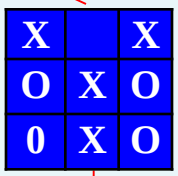


X-đi
Max

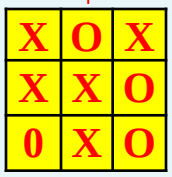


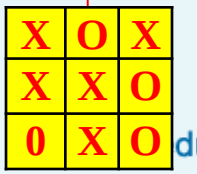


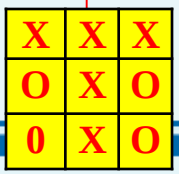




O-đi
Min







Cài đặt giải thuật cắt tỉa alpha – beta định trị cây trò chơi

```
float cat_tia(NodeType Q, ModeType mode,
      float Vp) {
      NodeType C;  /*C là một nút con của nút
      Q*/
      float Vq;
      /*Vq là giá trị tạm của Q, sau khi tất cả
      các con của nút Q đã xét hoặc bị cắt tỉa thì
      Vq là giá trị của nút Q*/
      if (is_leaf(Q)) return Payoff(Q);
      /* Khởi tạo giá trị tạm cho Q */
      if (mode == MAX) Vq = -∞;
      else Vq = ∞;
      /*Xét các con của Q, mỗi lần xác định
      được giá trị của một nút con của Q, ta
      phải đặt lại giá trị tạm Vq và so sánh với
      Vp để có thể cắt tỉa hay không*/
```

```
Xét C là con trái nhất của Q;
while (C là con của Q) {
   if (mode == MAX) {
      Vq = max(Vq, Cat_tia(C, MIN, Vq));
      if (Vp<=Vq) return Vq; /*cắt tỉa tất cả
      các con còn lại của Q*/
      } else {
        Vq = min(Vq, Cat_tia(C, MAX, Vq));
        if (Vp >= Vq) return Vq;
      }
return Vq;
}
```

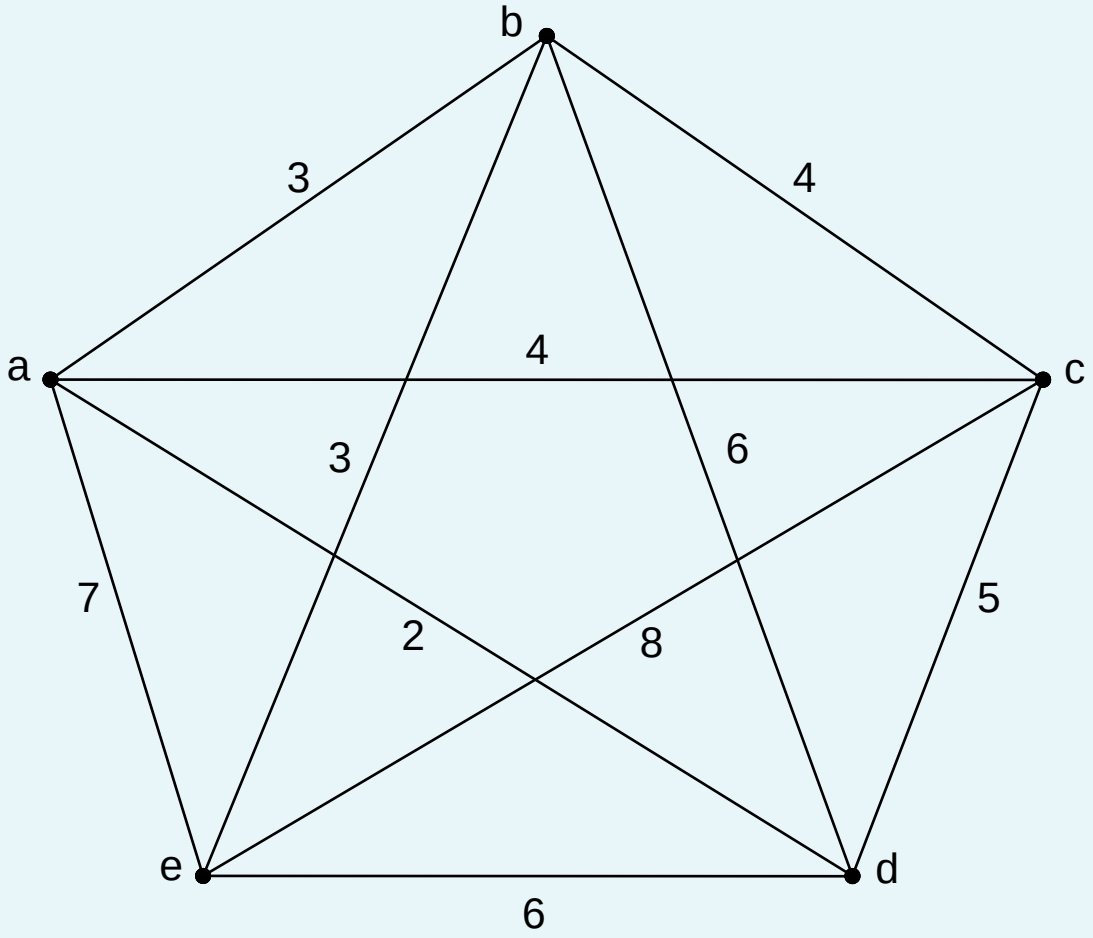
Kỹ thuật nhánh cận

- Nhánh cận là kỹ thuật xây dựng cây tìm kiếm phương án tối ưu, nhưng không xây dựng toàn bộ cây mà sử dụng giá trị cận để hạn chế bớt các nhánh.
- Cây tìm kiếm phương án có nút gốc biểu diễn cho tập tất cả các phương án có thể có, mỗi nút lá biểu diễn cho một phương án nào đó. Nút n có các nút con tương ứng với các khả năng có thể lựa chọn tập phương án xuất phát từ n. Kỹ thuật này gọi là **phân nhánh**.
- Vói mỗi nút trên cây ta sẽ xác định một giá trị cận. Giá trị cận là một giá trị gần với giá của các phương án. Với bài toán tìm min ta sẽ xác định cận dưới còn với bài toán tìm max ta sẽ xác định cận trên. Cận dưới là giá trị nhỏ hơn hoặc bằng giá của phương án, ngược lại cận trên là giá trị lớn hơn hoặc bằng giá của phương án.

Kỹ thuật nhánh cận: bài toán TSP

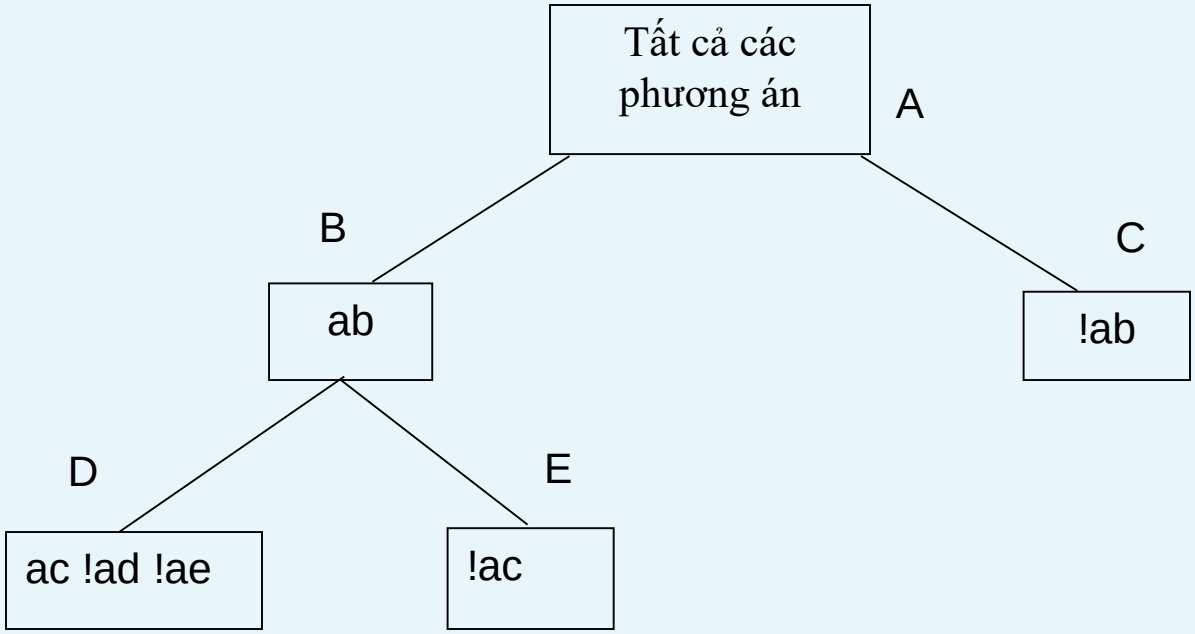
- Cây tìm kiếm phương án là **cây nhị phân** trong đó:
- Nút gốc là nút biểu diễn cho cấu hình gồm tất cả các phương án.
- Con trái biểu diễn cho cấu hình gồm tất cả các phương án chứa một cạnh nào đó, con phải biểu diễn cho cấu hình gồm tất cả các phương án không chứa cạnh đó (các cạnh để xét phân nhánh được thành lập tuân theo một thứ tự nào đó, chẳng hạn thứ tự từ điển).
- Mỗi nút sẽ kế thừa các thuộc tính của tổ tiên của nó và có thêm một thuộc tính mới (chứa hay không chứa một cạnh nào đó).
- Nút lá biểu diễn cho một cấu hình chỉ bao gồm một phương án.
- Để quá trình phân nhánh mau chóng tới nút lá, tại mỗi nút ta cần có một quyết định bổ sung dựa trên nguyên tắc là mọi đỉnh trong chu trình đều có cấp 2 và không tạo ra một chu trình thiếu.

Kỹ thuật nhánh cận: bài toán TSP – ví dụ:

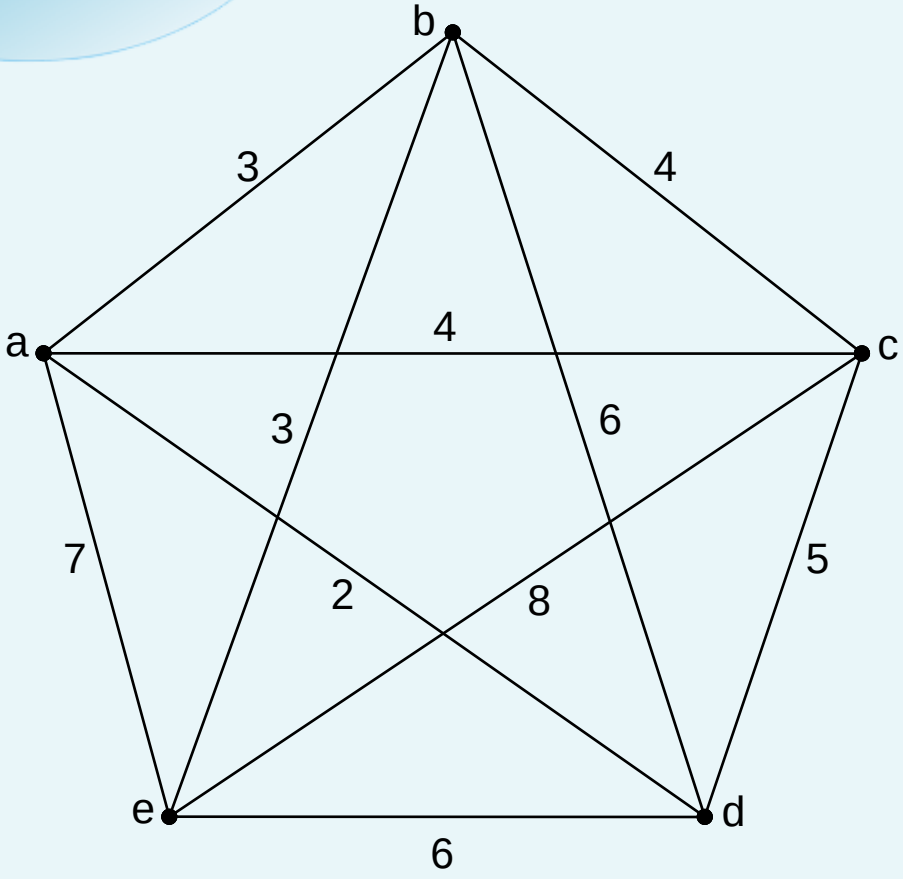


Kỹ thuật nhánh cận: bài toán TSP- Phân nhánh

Các cạnh theo thứ tự từ điển để xét là: ab, ac, ad, ae, bc, bd, be, cd, ce và de.



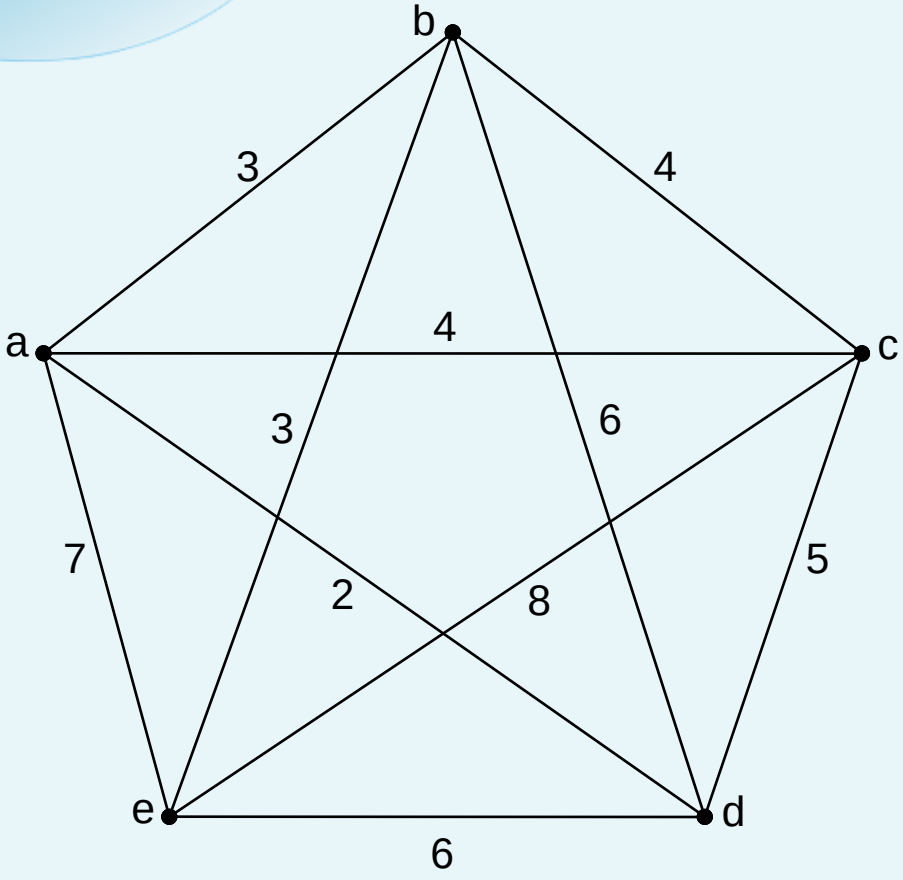
Tính cận dưới cho nút gốc A



- Đỉnh a chọn ad = 2, ab = 3
- Đỉnh b chọn ba = 3, be = 3
- Đỉnh c chọn ca = 4, cb = 4
- Đỉnh d chọn da = 2, dc = 5
- Đỉnh e chọn eb = 3, ed = 6

Tổng độ dài các cạnh được chọn là 35, cận dưới của nút gốc A là 35/2 = 17.5

Tính cận dưới cho nút D (ab, ac, !ad, !ae)

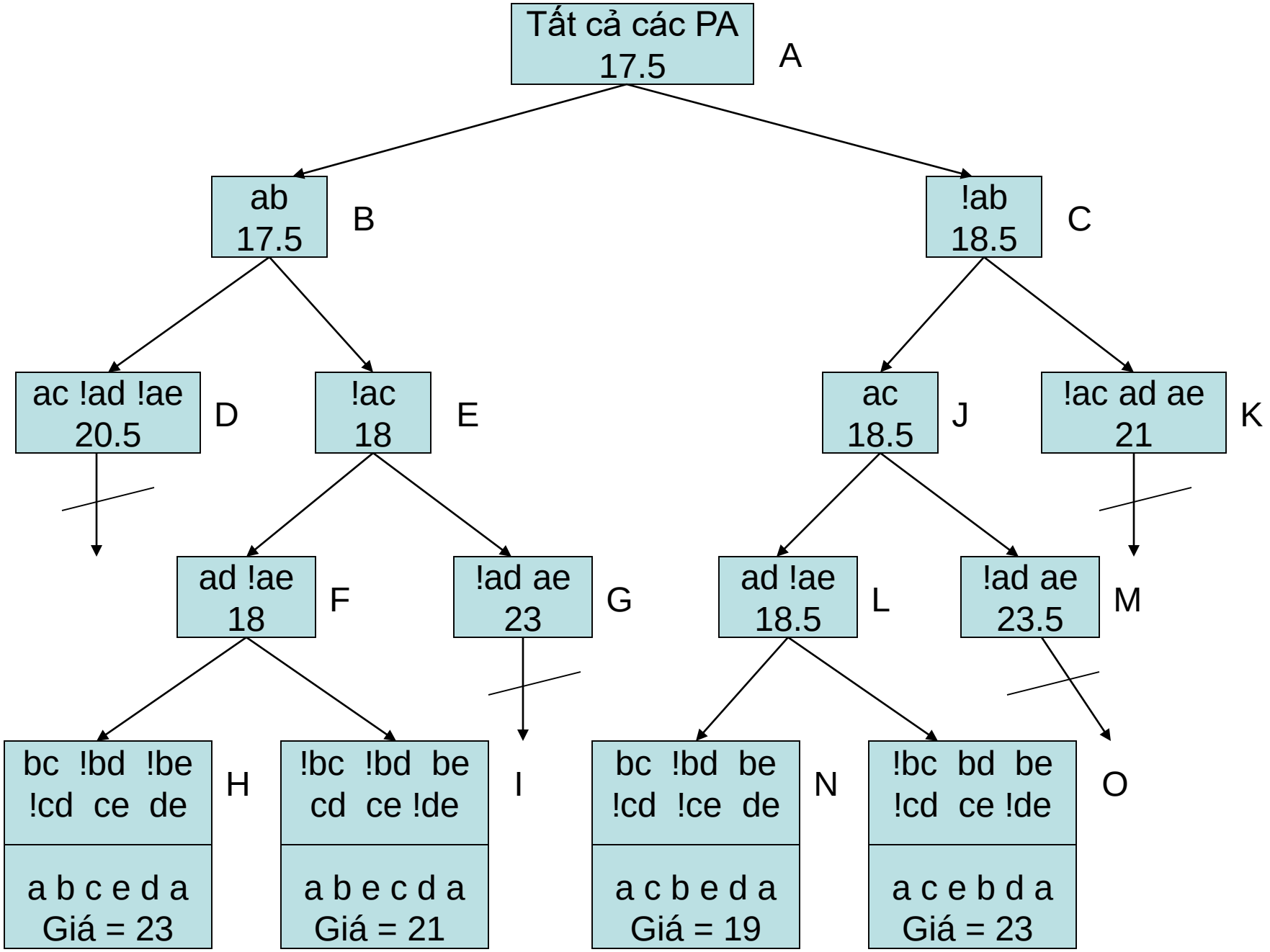


- Đỉnh a chọn ab = 3, ac = 4, do hai cạnh này buộc phải chọn.
- Đỉnh b chọn ba = 3, be = 3
- Đỉnh c chọn ca = 4, cb = 4
- Đỉnh d chọn de = 6, dc = 5, do không được chọn da nên ta phải chọn de.
- Đỉnh e chọn eb = 3, ed = 6

  Tổng độ dài các cạnh được chọn là 41, cận dưới của nút D là 41/2 = 20.5

Áp dụng kỹ thuật nhánh cận cho bài toán TSP

- Xây dựng nút gốc, tính cận dưới cho nút gốc.
- Sau khi phân nhánh cho mỗi nút, ta tính cận dưới cho cả hai con.
- Nếu cận dưới của một nút con >= giá nhỏ nhất tạm thời của một phương án đã được tìm thấy thì ta không cần xây dựng các cây con cho nút này nữa (Ta gọi là cắt tỉa các cây con của nút đó).
- Nếu cả hai con đều có cận dưới nhỏ hơn giá nhỏ nhất tạm thời của một phương án đã được tìm thấy thì nút con nào có cận dưới nhỏ hơn sẽ được ưu tiên phân nhánh trước.
- Mỗi lần quay lui để xét nút con chưa được xét của một nút ta phải xem xét lại nút con đó để có thể cắt tỉa các cây của nó hay không vì có thể một phương án có giá nhỏ nhất tạm thời vừa được tìm thấy.
- Sau khi tất cả các con đã được phân nhánh hoặc bị cắt tỉa thì phương án có giá nhỏ nhất trong các phương án tìm được là phương án cần tìm.
- Trong quá trình xây dựng cây có thể ta đã xây dựng được một số nút lá, như ta biết mỗi nút lá biểu diễn cho một phương án. Giá nhỏ nhất trong số các giá của các phương án này được gọi là giá nhỏ nhất tạm thời.


Tất cả các PA
17.5
A
ab
17.5
B
!ab
18.5
C
ac !ad !ae
20.5
D
!ac
18
E
ac
18.5
J
!ac ad ae
21
K
ad !ae
18
F
!ad ae
23
G
ad !ae
18.5
L
!ad ae
23.5
M
bc !bd !be
!cd ce de
a b c e d a
Giá = 23
H
!bc !bd be
cd ce !de
a b e c d a
Giá = 21
I
bc !bd be
!cd !ce de
a c b e d a
Giá = 19
N
!bc bd be
!cd ce !de
a c e b d a
Giá = 23
O

Kỹ thuật nhánh cận: BTcái ba lô

- Danh sách các đồ vật được sắp xếp theo thứ tự giảm của đơn giá
- Nút gốc biểu diễn cho trạng thái ban đầu của ba lô, ở đó ta chưa chọn một vật nào. Tổng giá trị được chọn TGT = 0. Cận trên của nút gốc CT = W * Đơn giá lớn nhất.
- Nút gốc sẽ có các nút con tương ứng với các khả năng chọn đồ vật có đơn giá lớn nhất. Với mỗi nút con ta tính lại các thông số:
  - TGT = TGT (của nút cha) + số đồ vật được chọn * giá trị mỗi vật.
  - W = W (của nút cha) - số đồ vật được chọn * trọng lượng mỗi vật.
  - CT = TGT + W * Đơn giá của vật sẽ xét kế tiếp.

Kỹ thuật nhánh cận: BTcái ba lô

- Trong các nút con, ta sẽ ưu tiên phân nhánh cho nút con nào có cận trên lớn hơn trước. Các con của nút này tương ứng với các khả năng chọn đồ vật có đơn giá lớn tiếp theo. Với mỗi nút ta lại phải xác định lại các thông số TGT, W, CT theo công thức đã nói trong bước 2.
- Lặp lại bước 3 với chú ý: đối với những nút có cận trên nhỏ hơn hoặc bằng giá lớn nhất tạm thời của một phương án đã được tìm thấy thì ta không cần phân nhánh cho nút đó nữa (cắt bỏ).
- Nếu tất cả các nút đều đã được phân nhánh hoặc bị cắt bỏ thì phương án có giá lớn nhất là phương án cần tìm.

Kỹ thuật nhánh cận: BTcái ba lô - **Ví dụ**

Ta có một ba lô có trọng lượng là 37 và 4 loại đồ vật với trọng lượng và giá trị tương ứng được cho trong bảng bên dưới:

ĐV	TL	GT
A	15	30
B	10	25
C	2	2
D	4	6

ĐV	TL	GT	ĐG
B	10	25	2.5
A	15	30	2.0
D	4	6	1.5
C	2	2	1.0

ĐV	TL	GT	ĐG
B	10	25	2.5
A	15	30	2.0
D	4	6	1.5
C	2	2	1.0