Đệ qui và Nhánh cận

THUẬT TOÁN ỨNG DỤNG

Đỗ Phan Thuận
thuandp.sinhvien@gmail.com

Bộ môn Khoa Học Máy Tính, Viện CNTT & TT,
Trường Đại Học Bách Khoa Hà Nội.

Ngày 3 tháng 10 năm 2019

Các mô hình giải bài cơ bản

Mô hình giải bài một phương pháp xây dựng bài giải cho một loại bài toán riêng biệt

- Duyệt toàn bộ
- Chia để trị
- Quy hoạch động
- Tham lam

Mỗi mô hình ứng dụng cho nhiều loại bài toán khác nhau

- Phương pháp vạn năng *Duyệt toàn bộ* (Brute force – Exhaustive search)
  - bài toán yêu cầu tìm một đối tượng có đặc tính riêng (*loại bài toán*)
  - áp dụng mô hình *Duyệt toàn bộ*: duyệt qua tất cả các đối tượng, với mỗi đối tượng, kiểm tra xem nó có đặc tính cần tìm không, nếu có, dừng lại, nếu không, tiếp tục tìm

Duyệt toàn bộ

- Cho một tập hữu hạn các phần tử
- Yêu cầu tìm một phần tử trong tập thỏa mãn một số ràng buộc
  - hoặc tìm **tất cả** các phần tử trong tập thỏa mãn một số ràng buộc
- Đơn giản! Chỉ cần duyệt qua tất cả các phần tử trong tập, với mỗi phần tử thì kiểm tra xem nó có thỏa mãn các ràng buộc không
- Tất nhiên là cách này không hiệu quả...
- Nhưng nhớ là ta luôn tìm bài giải đơn giản nhất mà chạy trong giới hạn thời gian
- Duyệt toàn bộ luôn là mô hình giải bài đầu tiên bạn nên nghĩ đến khi giải một bài toán

Phân tích thời gian tính khấu trừ (Amortized time

Đối với các thuật toán duyệt toàn bộ, khi số lượng lời giải lên đến hàm mũ, ta cần đánh giá hiệu quả của thuật toán thông qua thời gian tính khấu trừ.

Thời gian tính khấu trừ là thời gian tính trung bình toàn bộ thuật toán mà một cấu hình lời giải được liệt kê ra. Như vậy đại lượng này được tính bởi tổng độ phức tạp thuật toán chia cho tổng số cấu hình lời giải của bài toán.

Bài toán ví dụ: Vito's family

- http://uva.onlinejudge.org/external/100/10041.html

Thuật toán quay lui

- Tư tưởng chính của thuật toán này là xây dựng dần các thành phần của cấu hình lời giải S bằng cách thử tất cả các khả năng có thể. Xuất phát từ trạng thái rỗng của lời giải.
- **Mô tả**: giả thiết cấu hình lời giải được mô tả bởi một bộ gồm n thành phần $x_1, x_2, \ldots, x_n$. Giả sử đã xác định được $i-1$ thành phần $x_1, x_2, \ldots, x_{i-1}$ (*gọi là lời giải bộ phận cấp* $i-1$). Bây giờ cần xác định thành phần x_i bằng cách thử tất cả các ứng viên có thể có nhờ các luật chuyển. Với mỗi ứng viên C, kiểm tra xem C có chấp nhận được hay không, xảy ra 2 khả năng:
  1. nếu chấp nhận C thì xác định x_i theo C; nếu $i = n$ thì ghi nhận lời giải mới, trái lại tiến hành việc xác định x_{i+1}
  2. nếu không có khả năng nào cho x_i thì quay lại bước trước để xác định lại x_{i-1}
- *Lưu ý*: ghi nhớ tại mỗi bước những khả năng nào đã thử để tránh trùng lặp. Các thông tin này cần được lưu trữ theo cơ cấu stack (vào sau ra trước - LIFO)

Sơ đồ chung

Bước xác định x_i có thể được diễn tả qua thủ tục được tổ chức đệ quy dưới đây:

```
void Try(int i) {
  foreach (ung vien duoc chap nhan C) {
       <update cac bien trang thai>
       <ghi nhan x[i] moi theo C>
       if (i==n) <ghi nhan mot loi giai>
       else Try(i+1)
       <tra cac bien ve trang thai cu>
  }
}
```

Quá trình tìm kiếm lời giải theo thuật toán quay lui có thể được mô tả bởi cây tìm kiếm lời giải (Vẽ cây)

Liệt kê nhị phân

1. Liệt kê các xâu nhị phân độ dài n

```
1  void Try(int k) {
2     int i;
3     for (i=0; i<=1; i++) {
4        a[k] = i;
5        if (k==n) <ghi nhan mot cau hinh>
6        else Try(k+1);
7     }
8  }
```

BTVN: Viết chương trình trên máy tính và phân tích độ phức tạp khấu trừ của thuật toán

Liệt kê hoán vị

2. Liệt kê các hoán vị của n phần tử

```
1  void Try(int k) {
2     int i;
3     for (i=1; i<=n; i++)
4        if (!d[i]) {
5           a[k] = i;
6           d[i] = 1;
7           if (k==n) <ghi nhan mot cau hinh>
8           else Try(k+1);
9           d[i] = 0;
10       }
11    }
```

BTVN: Viết chương trình trên máy tính và phân tích độ phức tạp khấu trừ của thuật toán

Liệt kê tổ hợp

3. Liệt kê các tổ hợp chập m của n phần tử $\{1, 2, \ldots, n\}$

```
1  void Try(int k) {
2    int i;
3    for (i= a[k-1]+1; i<=n-m+k; i++) {
4      a[k] = i;
5      if (k==m) <ghi nhan mot cau hinh>
6      else Try(k+1);
7    }
8  }
```

BTVN: Viết chương trình trên máy tính và phân tích độ phức tạp khấu trừ của thuật toán

Bài toán chia kẹo

4. Liệt kê tất cả các cách chia M kẹo cho n em bé sao cho em bé nào cũng có kẹo
   - Đưa về bài toán liệt kê tất cả các nghiệm nguyên dương của phương trình tuyến tính
     $$x_1 + x_2 + \cdots + x_n = M$$
     với $(a_i)_{1 \le i \le n}$ và M và các số nguyên dương
   - Lời giải bộ phận $(x_1, x_2, \ldots, x_{k-1})$
   - $m = \sum_{i=1}^{k-1} x_i$
   - $A = n - k$
   - $\overline{M} = M - m - A$
   - Ứng viên x_k và $\{v \in \mathbb{Z} \mid 1 \le v \le \overline{M}\}$

Mã giả bài toán chia kẹo

Algorithm 1: TRY(i)

if $i = n$ **then**
    $\overline{M} \leftarrow M - f$;
    $\underline{M} \leftarrow M - f$;
else
    $\overline{M} \leftarrow M - f - (n - i)$;
    $\underline{M} \leftarrow 1$;
foreach $v = \underline{M}, \ldots, \overline{M}$ **do**
    $x_i \leftarrow v$;
    $f \leftarrow f + v$;
    if $i == n$ **then**
        printConfiguration();
    else
        TRY($i + 1$);
    $f \leftarrow f - v$;

Algorithm 2: MainLinearEquation(n, M)

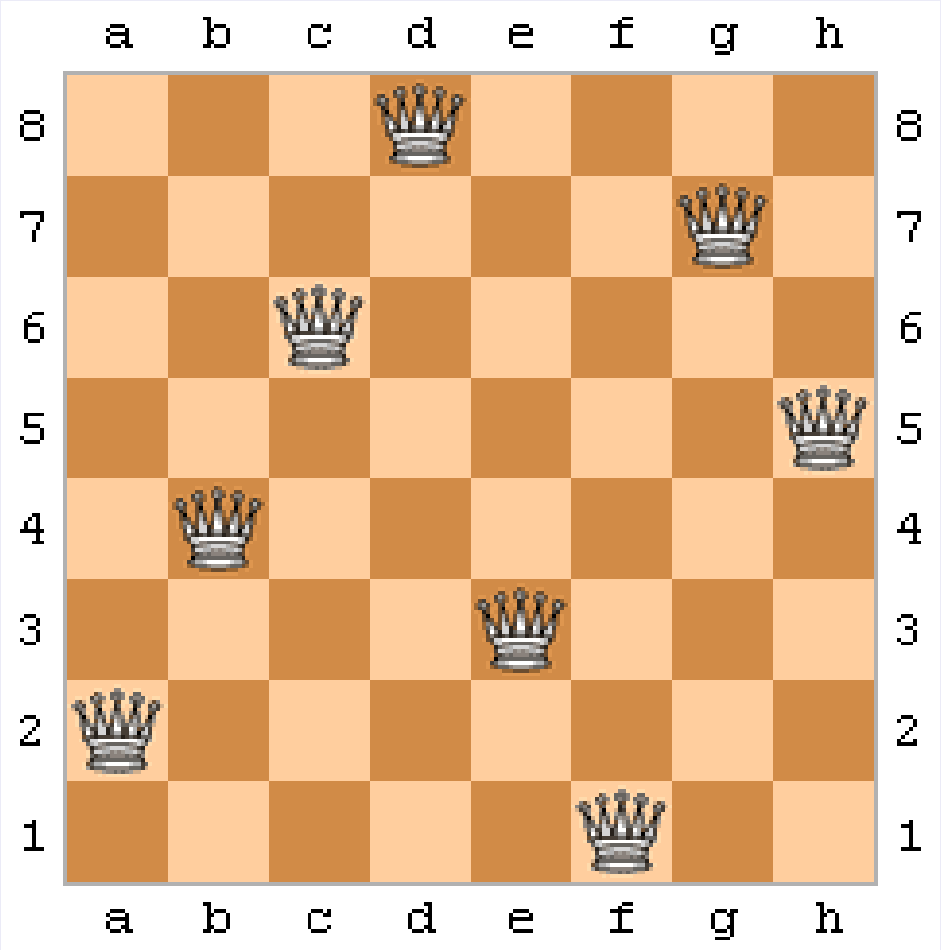
$f \leftarrow 0$;
TRY(1);

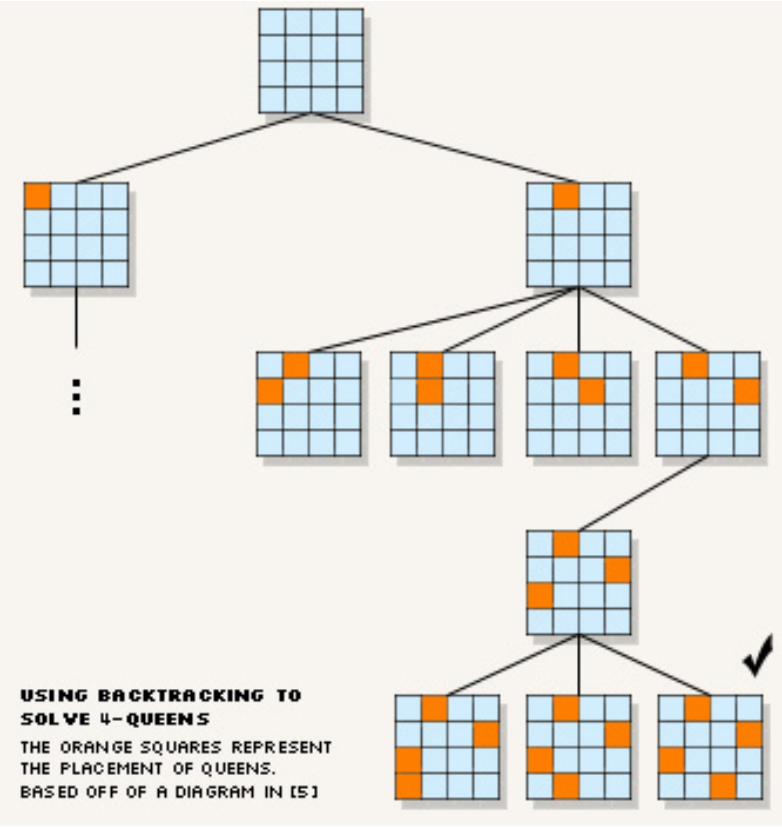
Bài toán chia kẹo: Số lượng lời giải

- Đầu tiên cho mỗi em bé một cái kẹo
- *Đáp số* : $\binom{M-1}{n-1}$

Bài toán xếp hậu

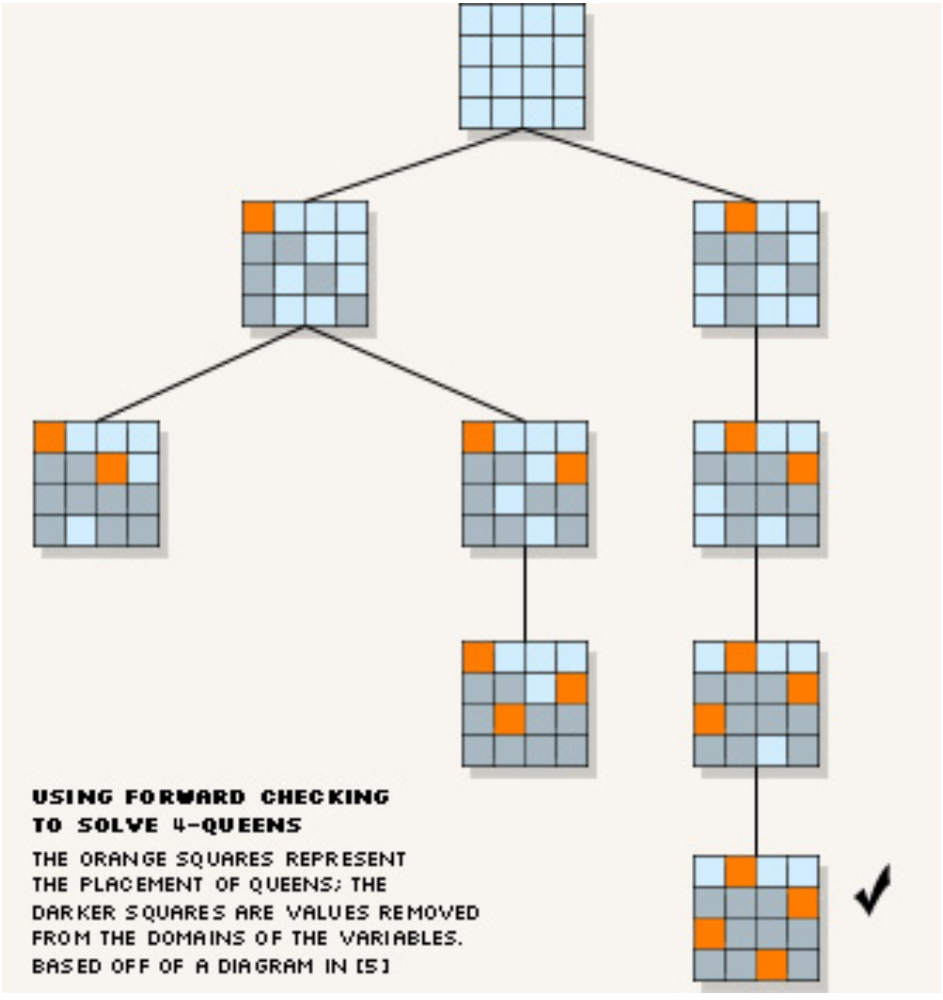
Liệt kê tất cả các cách xếp n quân Hậu trên bàn cờ $n \times n$ sao cho chúng không ăn được lẫn nhau.





USING BACKTRACKING TO SOLVE 4-QUEENS

THE ORANGE SQUARES REPRESENT THE PLACEMENT OF QUEENS. BASED OFF OF A DIAGRAM IN [5]



USING FORWARD CHECKING TO SOLVE 4-QUEENS

THE ORANGE SQUARES REPRESENT THE PLACEMENT OF QUEENS; THE DARKER SQUARES ARE VALUES REMOVED FROM THE DOMAINS OF THE VARIABLES. BASED OFF OF A DIAGRAM IN [5]

```
void Try(int i) {
  int j;
  for (j= 1; j<=n; j++)
    if (a(j) && b(i+j) && c(i-j)) {//chap nhan j
        x(i) = j;
        // ghi nhan trang thai moi
        a(j)=FALSE; b(i+j)=FALSE; c(i-j) = FALSE;
        if (i==n) OUTPUT;
        else Try(i+1);
        // tra lai trang thai cu
        a(j)=TRUE; b(i+j)= TRUE; c(i-j) = TRUE;
    }
}
```

n	3	4	7	8	9	10	11	12	13	14	15
H_n	0	2	40	92	352	724	2680	14,200	73,712	365,596	2,279,184
U_n	0	1	6	12	46	92	341	1,781	9,233	45,752	285,053

BTVN: Viết chương trình trên máy tính và phân tích độ phức tạp khấu trừ của thuật toán

Một số bài toán ví dụ

- Mã đi tuần: Cho bàn cờ $n \times n$ và một quân mã xuất phát tại vị trí (i, j). Hãy di chuyển quân mã trên bàn cờ sao cho có thể đi được toàn bộ các ô trên bàn cờ mà mỗi ô chỉ được qua 1 lần. Liệt kê tất cả khả năng có thể
- The Hamming Distance Problem
  http://uva.onlinejudge.org/external/7/729.html

Bài toán tối ưu tổ hợp

Chọn trong số tất cả các cấu hình tổ hợp chấp nhận được cấu hình có giá trị sử dụng tốt nhất.

Dạng tổng quát

$$f(x) \to \min(\max)$$

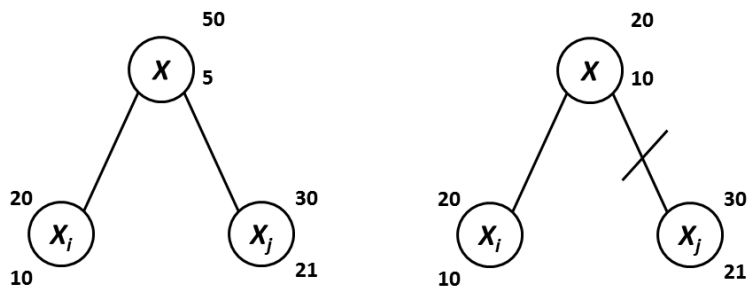
- $x \in D$ được gọi là một phương án,
- D gọi là tập các phương án của bài toán (thỏa mãn một số tính chất cho trước),
- hàm $f(x)$ gọi là hàm mục tiêu của bài toán,
- phương án $x^* \in D$ đem lại giá trị nhỏ nhất (lớn nhất) cho hàm mục tiêu được gọi là phương án tối ưu, khi đó $f^* = f(x^*)$ được gọi là giá trị tối ưu của bài toán.

Một số bài toán ứng dụng

- Bài toán người du lịch
- Bài toán cái túi
- Bài toán định tuyến xe (VRP)
- Bài toán lập lịch (Scheduling)
- Bài toán xếp thời khóa biểu (Timetabling)
- Bài toán đóng thùng (Bin Packing)
- Bài toán phân bổ tài nguyên (Resource allocations)
- ...

Thuật toán nhánh cận (TTNC)

- Là một phương pháp giải chủ yếu của bài toán tối ưu tổ hợp.
- Phân hoạch các phương án của bài toán thành 2 hay nhiều tập con được biểu diễn như các nút trên cây tìm kiếm.
- Tìm cách đánh giá cận nhằm loại bỏ những nhánh của cây tìm kiếm mà ta biết chắc là không chứa phương án tối ưu.
- Tình huống tồi nhất vẫn phải duyệt toàn bộ.



Mô hình bài toán MIN tổng quát

Bài toán

$$\min\{f(x) : x \in D\}$$

- D là tập hữu hạn phần tử:
  $D = \{x = (x_1, x_2, \ldots, x_n) \in A_1 \times A_2 \times \ldots \times A_n; x \text{ thoả mãn tính chất } P\}$,
- $A_1, A_2, \ldots, A_n$ là các tập hữu hạn,
- P là tính chất cho trên tích đề các $A_1 \times A_2 \times \ldots \times A_n$.

Nhánh cận

- Sử dụng thuật toán quay lui để xây dựng dần các thành phần của phương án.
- Gọi một bộ phận gồm k thành phần $(a_1, a_2, \ldots, a_k)$ xuất hiện trong quá trình thực hiện thuật toán sẽ được gọi là **phương án bộ phận** cấp k.

- Áp dụng TTNC trong trường hợp có thể tìm được một hàm g thoả mãn:

$$g(a_1, a_2, \ldots, a_k) \leq \min\{f(x) : x \in D, x_i = a_i, i = 1, 2, \ldots k\}$$

  với mọi lời giải bộ phận $(a_1, a_2, \ldots, a_k)$, và với mọi $k = 1, 2, \ldots$

- g được gọi là hàm **cận dưới**. Giá trị $g(a_1, a_2, \ldots, a_k)$ là cận dưới của phương án bộ phận $(a_1, a_2, \ldots, a_k)$.

Cắt nhánh

- Gọi $\bar{x}$ là giá trị hàm mục tiêu nhỏ nhất trong số các phương án đã duyệt, ký hiệu $\bar{f} = f(\bar{x})$. Ta gọi $\bar{x}$ là **phương án tốt nhất hiện có**, còn $\bar{f}$ là **kỷ lục**.

- Nếu: $g(a_1, a_2, \ldots, a_k) > \bar{f}$
  $\Rightarrow \bar{f} < g(a_1, a_2, \ldots, a_k) \leq \min\{f(x) : x \in D, x_i = a_i, i = 1, 2, \ldots k\}$

  $\Rightarrow$ tập con các phương án của bài toán $D(a_1, a_2, \ldots, a_k)$ chắc chắn không chứa phương án tối ưu.

  $\Rightarrow$ không cần phải phát triển phương án bộ phận $(a_1, a_2, \ldots, a_k)$

  $\Rightarrow$loại bỏ các phương án trong tập $D(a_1, a_2, \ldots, a_k)$ khỏi quá trình tìm kiếm.

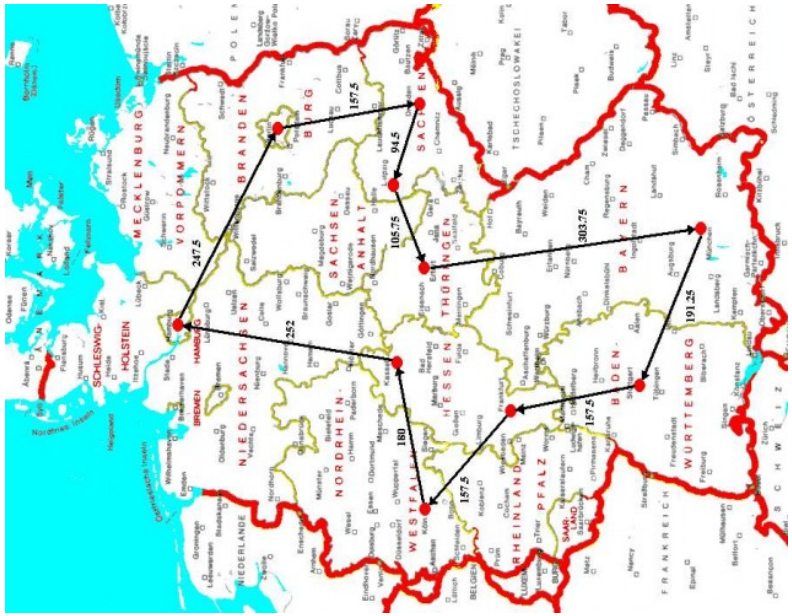
```
void TRY(int k) {
    /* Phát triển phương án bộ phận (a_1, a_2, ..., a_k)
    theo thuật toán quay lui có kiểm tra cận dưới */
    for (a_k ∈ A_k) {
        if <chấp nhận a_k> {
            x(k) = a_k;
            if (k == n) <Cập nhật kỷ lục>;
            else if (g(a_1, a_2, ..., a_k) ≤ f̄)
                    try(i + 1);
        }
    }
}

void Nhanh_Can(void) {
    f̄ = +∞;
    /* Nếu biết một phương án x nào đó thì có thể đặt f̄ = f(x̄). */
    TRY(1);
    if(f̄ < +∞) <f̄ là giá trị tối ưu, x̄ là phương án tối ưu>;
    else <bài toán không có phương án>;
}
```

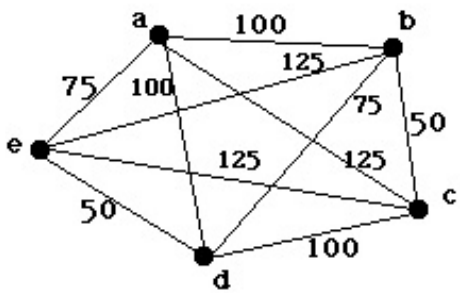
Nhận xét

- Nếu không có điều kiện cắt nhánh if $(g(a_1, a_2, \ldots, a_k) \leq \bar{f})$ thì thủ tục Try sẽ liệt kê toàn bộ các phương án của bài toán $\Rightarrow$ thuật toán duyệt toàn bộ.
- Việc xây dựng hàm g phụ thuộc vào từng bài toán cụ thể.
- Việc tính giá trị của g phải đơn giản hơn việc giải bài toán gốc.
- Giá trị của $g(a_1, a_2, \ldots, a_k)$ phải sát với giá trị tối ưu của bài toán gốc.

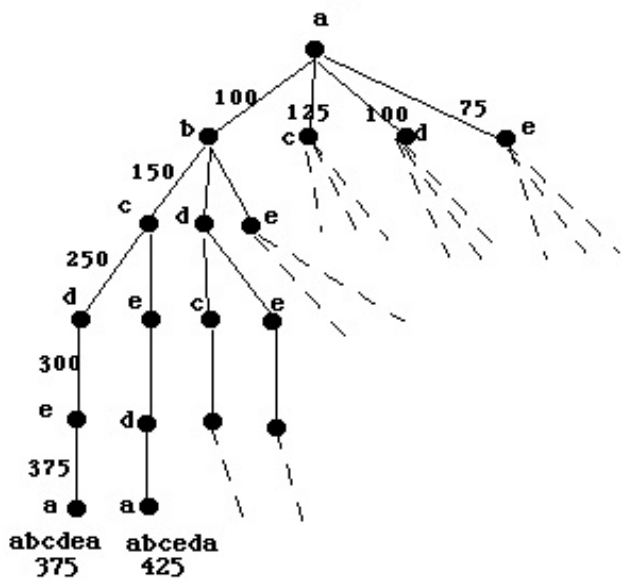
TTNC giải bài toán Người du lịch



An Instance of the
Traveling Salesman Problem



Search Space



Cố định thành phố xuất phát là T_1. Bài toán Người du lịch được đưa về bài toán: Tìm cực tiểu của hàm

$$f(x_2, x_3, \ldots, x_n) = c[1, x_2] + c[x_2, x_3] + \ldots + c[x_{n-1}, x_n] + c[x_n, x_1] \to \min$$

Gọi:

$$c_{\min} = \min \{c[i, j], i, j = 1, 2, \ldots, n, i \neq j\}$$

là chi phí đi lại nhỏ nhất giữa các thành phố.
Giả sử ta đang có phương án bộ phận $(u_1, u_2, \ldots, u_k)$ tương ứng với hành trình bộ phận qua k thành phố:

$$T_1 \to T(u_2) \to \ldots \to T(u_{k-1}) \to T(u_k).$$

với chi phí phải trả theo hành trình bộ phận này là

$$\sigma = c[1, u_2] + c[u_2, u_3] + \ldots + c[x_{k-1}, u_k].$$

Do chi phí phải trả cho việc đi qua mỗi một trong số $n - k + 1$ đoạn đường còn lại đều không nhiều hơn $c_{\min} \Rightarrow$ **cận dưới** cho phương án bộ phận $(u_1, u_2, \ldots, u_k)$:

$$g(u_1, u_2, \ldots, u_k) = \sigma + (n - k + 1)c_{\min}$$

Algorithm 3: MainSimpleBBTSP(n, c)

Input: n, c: số lượng thành phố n và ma trận chi phí c
x, f^*, f, *visited* là các biến toàn cục
Output: Chi phí của hành trình tiết kiệm nhất
foreach $v = 1, \ldots, n$ **do**
    $visited[v] \leftarrow FALSE$;
$f^* \leftarrow \infty$;
$f \leftarrow 0$;
$x_1 \leftarrow 1$;
$visited[x_1] \leftarrow TRUE$;
TRY(2);
return f^*;

Algorithm 4: TRY(k)

Input: k: chỉ số thành phố thứ k được thăm
n, c, x, f^*, f, *visited* là các biến toàn cục
Output: Mở rộng phương án bộ phận hiện tại $x_1, \ldots, x_{k-1}$ bằng việc bổ sung thêm giá trị x_k

```
foreach v = 1, ..., n do
    if visited[v] = FALSE then
        x_k ← v;
        visited[v] ← TRUE;
        f ← f + c(x_{k-1}, x_k);
        if k = n then
            if f + c(x_n, x_1) < f* then
                f* ← f + c(x_n, x_1);
        else
            z ← f + (n − k + 1) ∗ cmin;
            if z < f* then
                TRY(k + 1);
        f ← f − c(x_{k-1}, x_k);
        visited[v] ← FALSE;
```