Một số mô hình thuật toán

Brute force, greedy, branch and bound, divide and conquer, dynamic programming

Nội dung

- Bài toán tối ưu hóa tổ hợp
- Thuật toán vét cạn
- Nhánh và cận
- Thuật toán tham lam
- Chia để trị
- Quy hoạch động

Bài toán tối ưu hóa

- Bài toán tối ưu hóa
  - Tìm kiếm lời giải tốt nhất trong những lời giải khả thi
  - Nếu đầu vào là rời rạc → Bài toán tối ưu tổ hợp
- Bài toán tối ưu tổ hợp
  - Không gian tìm kiếm là hữu hạn hoặc vô hạn đếm được
  - Thỏa mãn tập ràng buộc C
  - Với mục đích tối ưu hàm mục tiêu, f(x) → min (max)
- Phương án giải
  - Có thể vét cạn hết không gian lời giải khả thi
  - Xây dựng từng bước lời giải
  - Giải gần đúng (xấp xỉ)

Thuật toán vét cạn

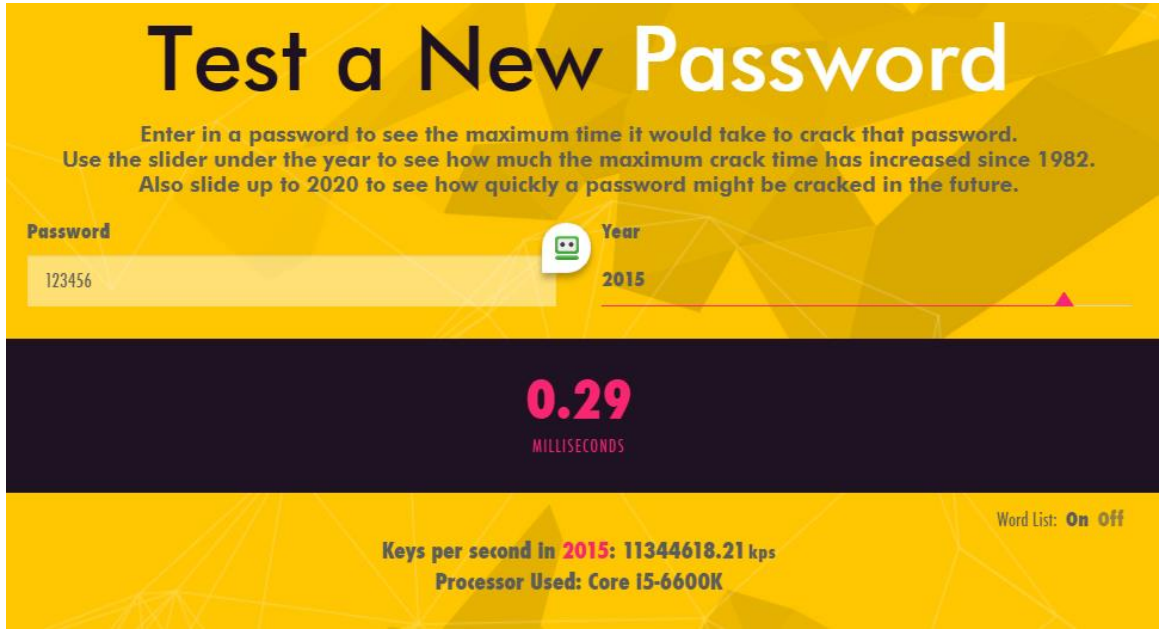
- Thuật toán vét cạn: brute-force search hoặc exhaustive search
  - Là một dạng của thuật toán dạng “sinh và test” – sinh ra phương án và kiểm tra lại xem có đáp ứng ràng buộc của bài toán
  - Đây là cách thiết kế thuật toán đơn giản nhất (chỉ cần nhìn vào phát biểu bài toán là tạo ra thuật toán được), dựa vào khả năng tính toán của máy tính
  - Hiệu quả của thuật toán không cao, nhất là khi kích thước đầu vào lớn
  - Một số bài toán phức tạp không thể giải bằng phương pháp vét cạn được (không gian tìm kiếm phương án có thể quá lớn, vượt khả năng tính toán của máy tính)
  - Có thể kết hợp thêm với một số kỹ thuật khác (heuristics methods) để giảm bớt không gian cần tìm kiếm lời giải
  - Dùng để giải các bài toán khi mà yêu cầu về thời gian chạy không phải là yếu tố đáng quan tâm (VD. chỉ cần tìm ra lời giải là được)

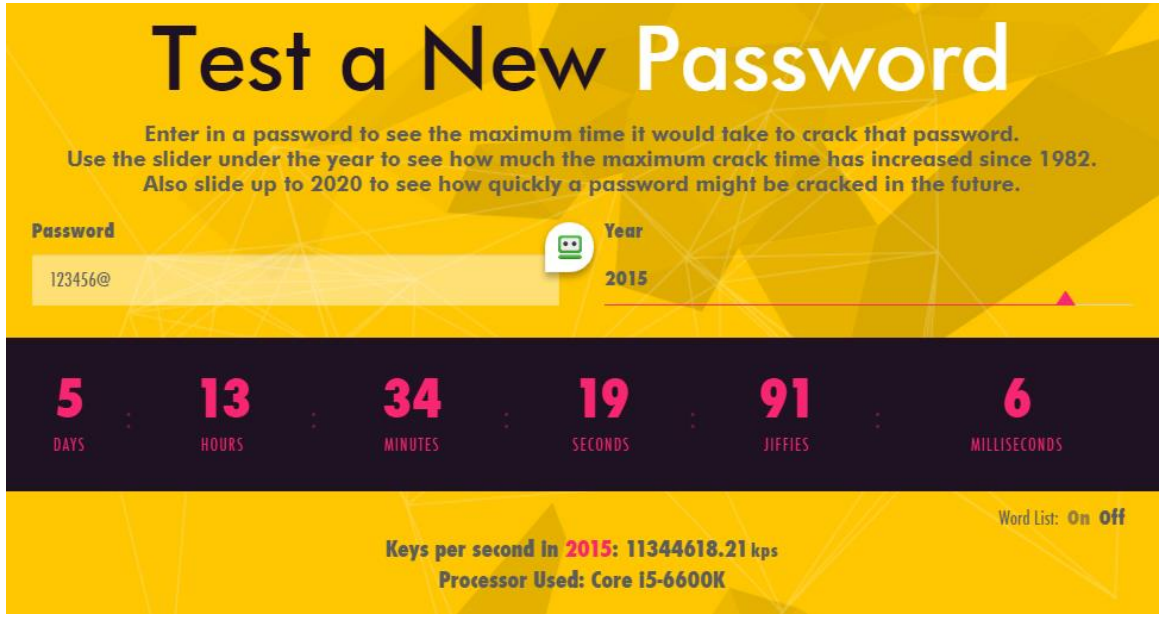
Thuật toán vét cạn

- Thuật toán vét cạn
  - Trong thuật toán luôn có 2 pha là sinh lựa chọn và kiểm tra
  - Có thể sinh lựa chọn là 1 phương án đầy đủ, hoặc từng bước sinh 1 phần của phương án (VD. thuật toán back tracking)
  - Pha kiểm tra:
    - xem phương án sinh ra có thỏa mãn là một lời giải của bài toán hay không,
    - thường dựa ngay chính vào mô tả ban đầu
    - Hoặc dùng chính hàm đích cần đạt được của bài toán làm hàm kiểm tra
    - Bước kiểm tra đôi khi có thể gộp luôn vào quá trình sinh

Thuật toán vét cạn

- Bài toán 1. Dò mật khẩu *******
- Mật khẩu
  - là chuỗi các ký tự (thường bao gồm ký tự đặc biệt và số)
  - Độ dài tối thiểu 6-8 ký tự
  - Mật khẩu càng dài, càng khó đoán nhưng cũng khó nhớ
  - Thông thường là dò bằng cách thử hết các khả năng, tuy nhiên có thể kết hợp với các từ điển các password hay dùng để cải thiện thời gian
  - Thường kết hợp song song chạy trên nhiều CPU





https://www.grc.com/haystack.htm

We use an AI password cracker called **PassGAN** to run through a list of 15,680,000 passwords. Here is what we found:

Time It Takes Using AI to Crack Your Password [2023]

# OF CHARACTER	Numbers Only	Lowercase Letters	Upper & Lowercase Letters	Numbers, Upper & Lowercase Letters	Numbers, Upper & Lowercase Letters, Symbols
4	Instantly	Instantly	Instantly	Instantly	Instantly
5	Instantly	Instantly	Instantly	Instantly	Instantly
6	Instantly	Instantly	Instantly	Instantly	4 Seconds
7	Instantly	Instantly	22 Seconds	42 Seconds	6 Minutes
8	Instantly	3 Seconds	19 Minutes	48 Minutes	7 Hours
9	Instantly	1 Minutes	11 Hours	2 Days	2 Weeks
10	Instantly	1 Hours	4 Weeks	6 Months	5 Years
11	Instantly	23 Hours	4 Years	38 Years	356 Years
12	25 Seconds	3 Weeks	289 Years	2K Years	30K Years
13	3 Minutes	11 Months	16K Years	91K Years	2M Years
14	36 Minutes	49 Years	827K Years	9M Years	187M Years
15	5 Hours	890 Years	47M Years	613M Years	14Bn Years
16	2 Days	23K Years	2Bn Years	26Bn Years	1Tn Years
17	3 Weeks	812K Years	539.72M Years	2Tn Years	95Tn Years
18	10 Months	22M Years	7.23Bn Years	96Tn Years	6Qn Years

HOME SECURITY HEROES

Thuật toán vét cạn

- Bài toán 2. Thuật toán sắp xếp lựa chọn
  - Lần lượt chọn phần tử lớn nhất trong dãy hiện tại, đổi chỗ với phần tử ở cuối dãy
  - Lặp lại bước trên nhưng chỉ xét dãy mới là n-1 phần tử đầu
- Nhận xét
  - Thuật toán xây dựng lời giải từng bước từ lời giải bộ phận
  - Bước kiểm tra đã nằm sẵn trong bước xây dựng phương án lựa chọn

```
selectionSort(A[0..n-1])
{
   for i=n to 2 do
      min = 0
      for j=0 to i-1 do
         if (A[j]>A[min])  min = j
      SWAP(A[min], A[i-1])
}
```

$$T(n) = O(n^2)$$

VD. 1, 5, 12, 3, 2, 4 ➔ 1, 5, 4, 3, 2,12 ➔ 1, 2, 4, 3, 5, 12 ➔ 1, 2, 3, 4, 5, 12 ➔ 1, 2, 3, 4, 5, 12 ➔ 1, 2, 3, 4, 5, 12

Thuật toán vét cạn

- Bài toán 3. Tính giá trị đa thức bậc n: $P_n(x) = a_n x^n + a_{n-1} x^{n-1} + .. + a_1 x + a_0$

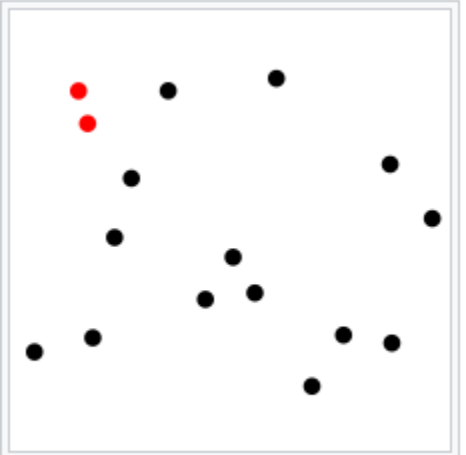
```
Algorithm1(A[0..n], x)
P = 0
For i=n to 0 do
   power = 1;
   for j=1 to i do
        power = power * x
   P = P+a[i]*power
Return P
```

$T_1(n) = O(n^2)$

```
Algorithm2(A[0..n], x)
P = a[n]
For i=n-1 to 0 do
    P = P*x+a[i]
Return P
```

$T_2(n) = O(n)$

Thuật toán vét cạn


Closest pair of points shown in red

- Bài toán 3. cặp điểm gần nhau nhất trong không gian
  - Cho danh sách n điểm trong không gian 2 chiều (tọa độ x,y)
  - Hãy đưa ra cặp điểm gần nhau nhất trong danh sách

Thử tất cả các cặp điểm có thể $T(n) = O(n^2)$

Giải pháp tốt hơn?

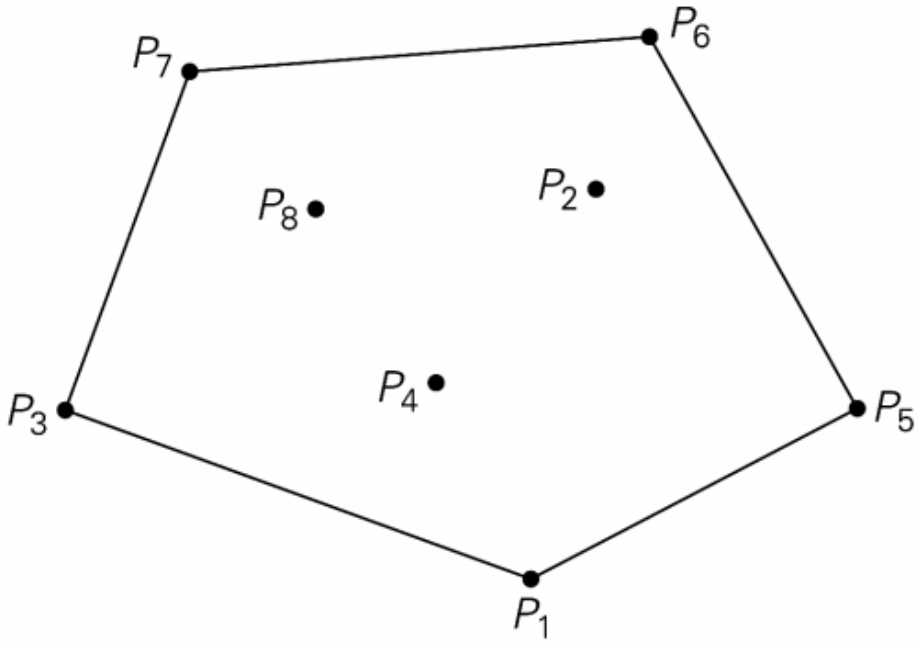
```
ClosestPair(double P[0..n-1])
{
   dmin = ∞
   for i=0 to n-2 do
      for j = i to n-1 do
            dis = Distance(P[i],P[j])
            if(dis<dmin)
                  dmin=dis
                  p1 =P[i], p2 = P[j]
   return dmin, p1, p2
}
```

Thuật toán vét cạn

- Bài toán 4. Tìm bao lồi – convex hull
  - Bao lồi của 1 tập n điểm trong không gian 2 chiều là 1 đa giác nhỏ nhất chứa n điểm bên trong
  - Bao lồi nhỏ nhất sẽ là 1 tập con của n điểm ban đầu

Ý tưởng:

- Với mỗi 1 cặp điểm p1, p2 sẽ tạo ra 1 đường thẳng
- Check xem các điểm còn lại trong tập n điểm có nằm về cùng 1 phía với đường thẳng tạo bởi p1, p2 không



https://en.wikipedia.org/wiki/Convex_hull

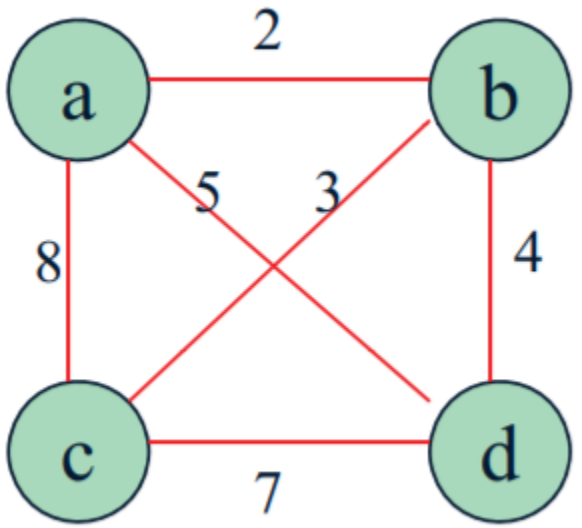
Thuật toán vét cạn

- Bài toán 5: người bán hàng – Traveling Salesperson Problem
  - Cho danh sách n thành phố với chi phí (khoảng cách) để đi lại giữa các cặp thành phố đó
  - Hãy xây dựng hành trình di chuyển giữa các thành phố sao cho mỗi thành phố chỉ được đến 1 lần trước khi trở lại thành phố xuất phát

Vét cạn: thử tất cả các phương án đi có thể

$$T(n) = O(n!)$$

Tour	Cost
a→b→c→d→a	2+3+7+5 = 17
a→b→d→c→a	2+4+7+8 = 21
a→c→b→d→a	8+3+4+5 = 20
a→c→d→b→a	8+7+4+2 = 21
a→d→b→c→a	5+4+3+8 = 20
a→d→c→b→a	5+7+3+2 = 17



Thuật toán vét cạn

- Bài toán 6. Bài toán cái túi
  - Đầu vào n đồ vật với trọng lượng w_i và giá trị c_i, cái túi với tải trọng tối đa W
  - Tìm cách chất đồ vật vào túi sao cho không vượt quá tải trọng và tổng giá trị là lớn nhất
- Vét cạn: Duyệt hết các tập con của n đồ vật $T(n) = 2^n$
  - Thuật toán sinh các hoán vị của tập hợp
- Thuật toán hiệu quả hơn?

Subset	Total weight	Total value
{1}	2	$20
{2}	5	$30
{3}	10	$50
{4}	5	$10
{1,2}	7	$50
{1,3}	12	$70
{1,4}	7	$30
{2,3}	15	$80
{2,4}	10	$40
{3,4}	15	$60
{1,2,3}	17	not feasible
{1,2,4}	12	$60
{1,3,4}	17	not feasible
{2,3,4}	20	not feasible
{1,2,3,4}	22	not feasible

W=16

item	weight	value
1	2	$20
2	5	$30
3	10	$50
4	5	$10

Thuật toán vét cạn

	Job 1	Job 2	Job 3	Job 4
Person 1	9	2	7	8
Person 2	6	4	3	7
Person 3	5	8	1	8
Person 4	7	6	9	4

- Bài toán 7. Bài toán phân công công việc
  - Đầu vào là danh sách n công nhân, và n công việc cần được phân công
  - Mỗi công nhân lại có kỹ năng khác nhau nên thời gian hoàn thành mỗi công việc sẽ khác nhau, được cho bởi ma trận nxn
  - Hãy xây dựng phương án phân công sao cho tổng thời gian hoàn thành n việc là nhỏ nhất

Thuật toán vét cạn
- Sinh ra các phương án phân công có thể
- Tính tổng chi phí

$$T(n) = O(n!)$$

Assignment (col.#s)	Total Cost
1, 2, 3, 4	9+4+1+4=18
1, 2, 4, 3	9+4+8+9=30
1, 3, 2, 4	9+3+8+4=24
1, 3, 4, 2	9+3+8+6=26
1, 4, 2, 3	9+7+8+9=33
1, 4, 3, 2	9+7+1+6=23

Thuật toán nhánh cận - branch and bound

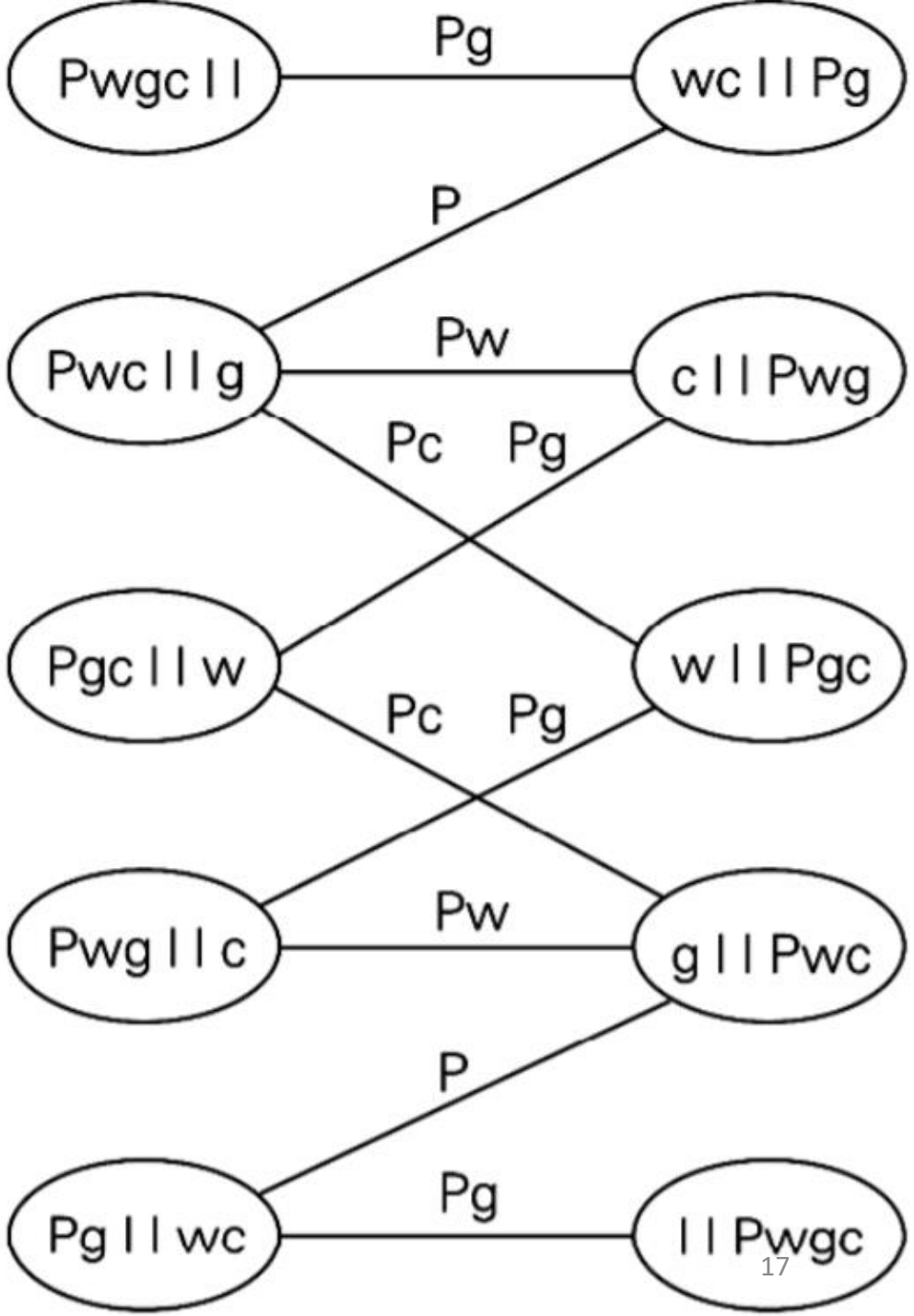
- **Thuật toán nhánh cận**
  - Là một cải tiến của thuật toán quay lui – back tracking
  - Áp dụng để giải các bài toán tối ưu tổ hợp
  - Thuật toán đi xây dựng lời giải từng bước, thường biểu diễn dưới dạng cây trạng thái, mỗi nút là 1 bộ phận của lời giải cuối cùng
  - Tại mỗi nút (lời giải bộ phận) sẽ tính toán 1 giá trị giới hạn dựa trên hàm mục tiêu cho các nút con, cháu (phần tiếp theo của lời giải) của nó.
  - Giá trị giới hạn – bound đó dùng để
    - Cắt tỉa các nhánh mà không tiềm năng (cắt bỏ các nhánh mà giá trị giới hạn này tồi hơn giá trị tốt nhất hiện có) → tránh phải duyệt hết các nhánh không tiềm năng
    - Dùng để định hướng quá trình duyệt qua cây trạng thái (VD. ưu tiên các nhanh tiềm năng –có giá trị bound tốt nhất trước)

Thuật toán nhánh cận - branch and bound

- Không gian trạng thái của bài toán
  - Chứa tất cả các trạng thái có thể
  - Thường được biểu diễn dưới dạng đồ thị hoặc cây
  - Trạng thái khởi đầu: là lời giải ban đầu của bài toán (chưa tối ưu)
  - Trạng thái đích: là lời giải cuối cùng cần đạt được
- Có các luật quy định về việc chuyển giữa các trạng thái
- Thuật toán có thể được hiểu dưới dạng 1 bài toán tìm đường đi từ trạng thái khởi đầu về trạng thái đích
- Phù hợp nhất với các bài toán giải đố, game,..

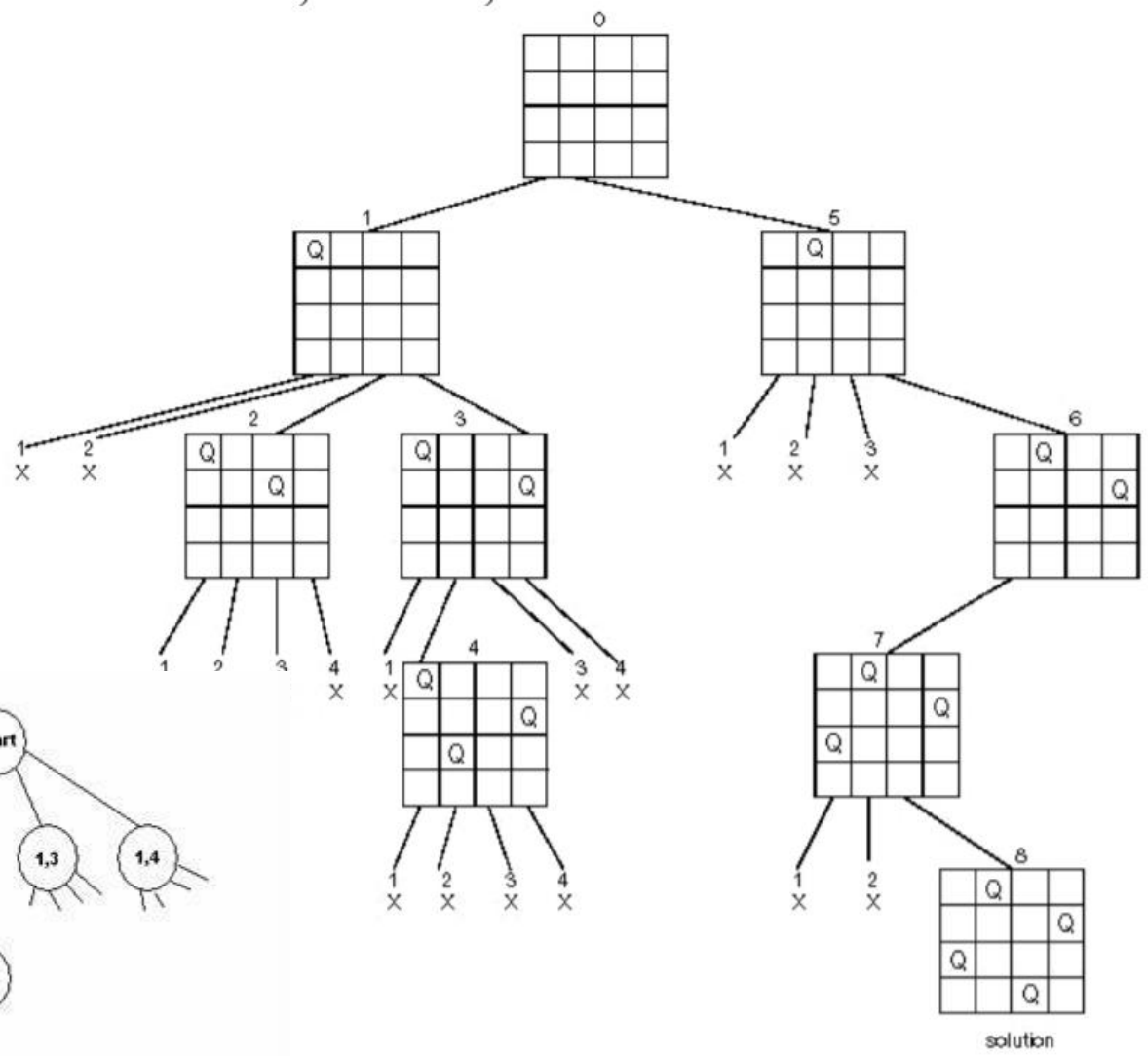
Thuật toán nhánh cận

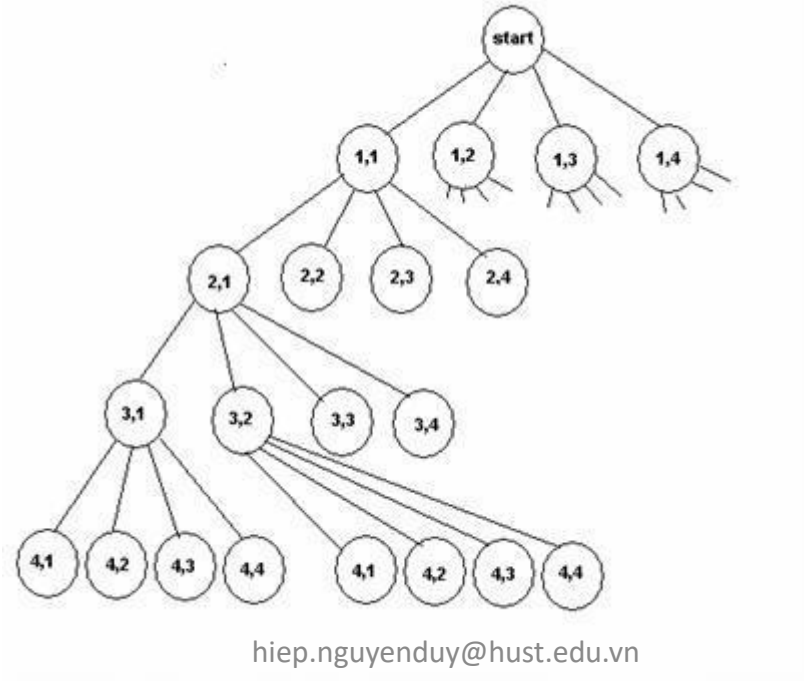
- Bài toán qua sông:
  - Người nông dân – P, Dê – G, chó sói – W và bao bắp cải – C
  - Mỗi lần qua sông, người nông dân chỉ chở được 1 thứ
  - Những thứ không được để cùng nhau mà không có mặt người nông dân: (G và W), (G và C) vì chúng sẽ ăn lẫn nhau
- Cây trạng thái của bài toán



Thuật toán nhánh cận

- Bài toán xếp hậu với bàn cờ 4x4
  - Mỗi quân hậu phải đứng trên 1 cột riêng, hàng riêng và đường chéo riêng
- Không gian trạng thái





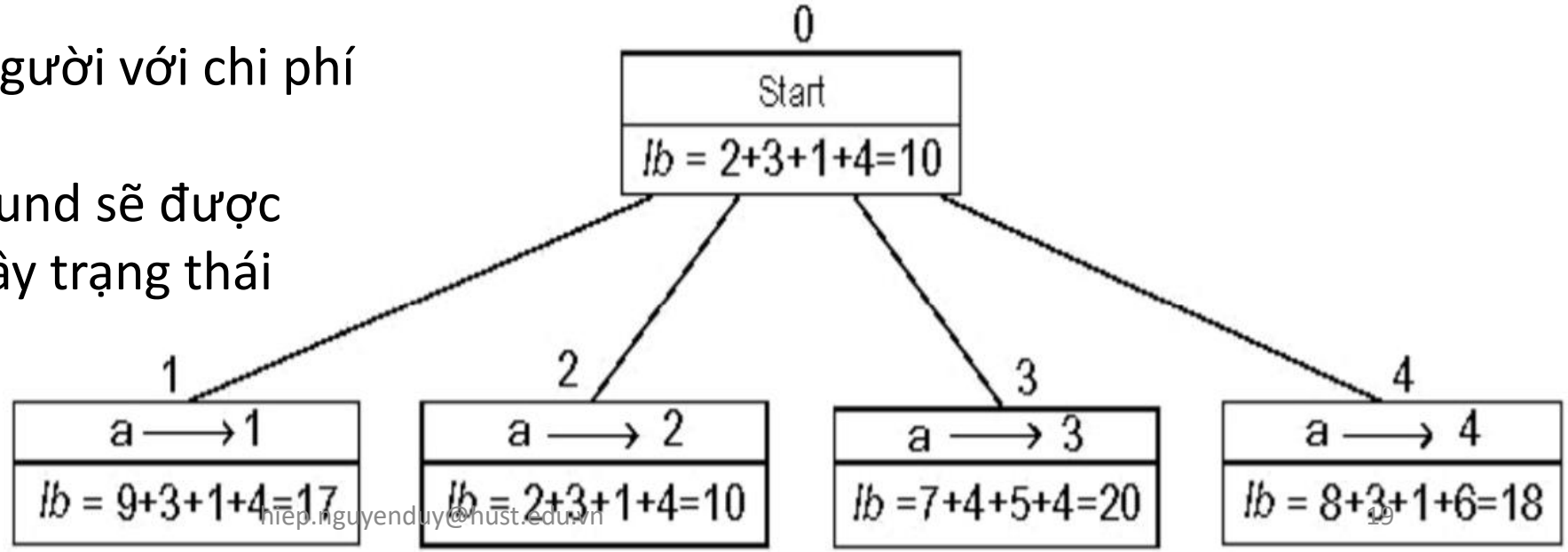
Thuật toán nhánh cận

	Job 1	Job 2	Job 3	Job 4
Person *a*	9	2	7	8
Person *b*	6	4	3	7
Person *c*	5	8	1	8
Person *d*	7	6	9	4

- Ví dụ 1. **Bài toán phân công công việc**
  - Mỗi người chỉ được phân 1 việc
  - Tổng chi phí để hoàn thành n việc là nhỏ nhất
- Cận dưới: 2 + 3 + 1 + 4 (or 5 + 2 + 1 + 4) Theo người hoặc theo việc

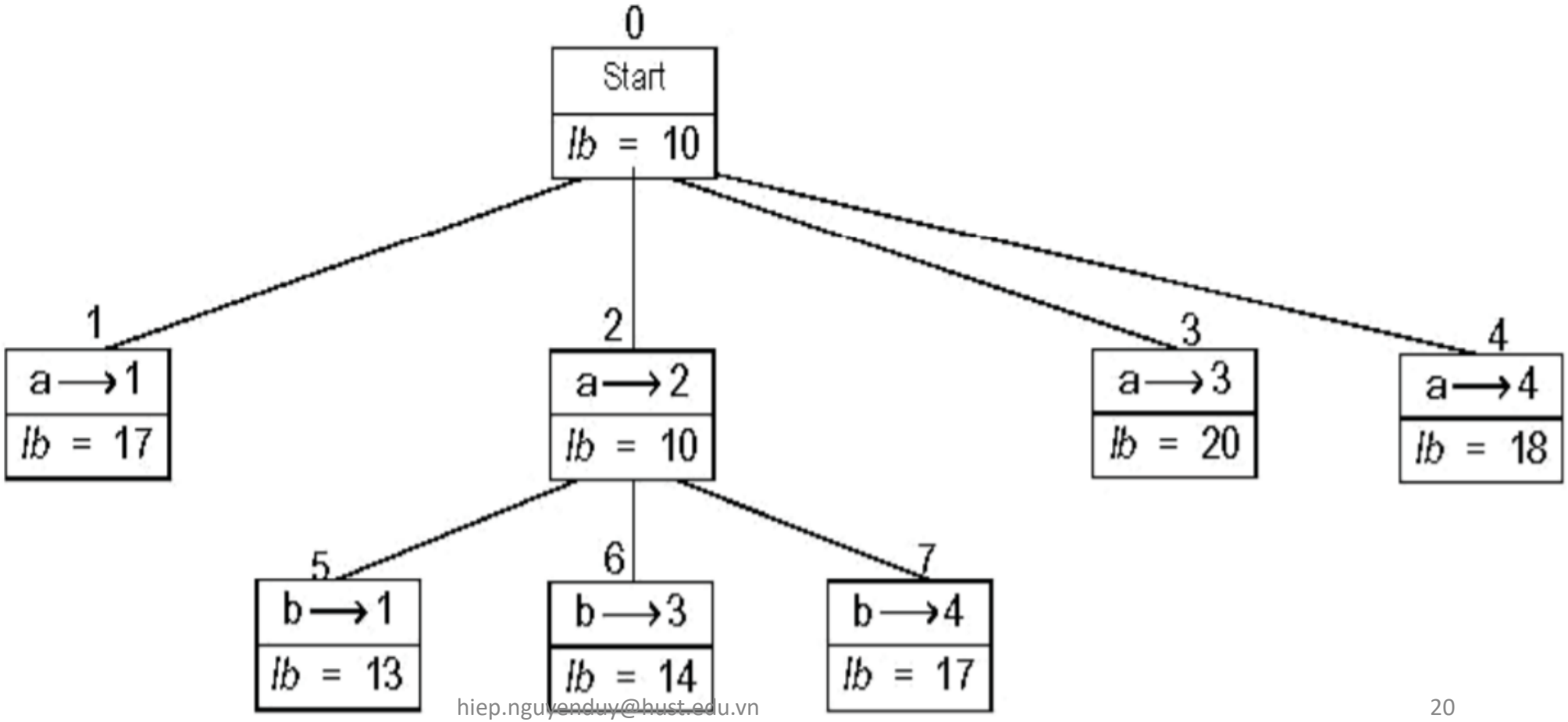
Bước 0 và 1 được tạo ra với thuật toán **best-first Branch-and-bound**

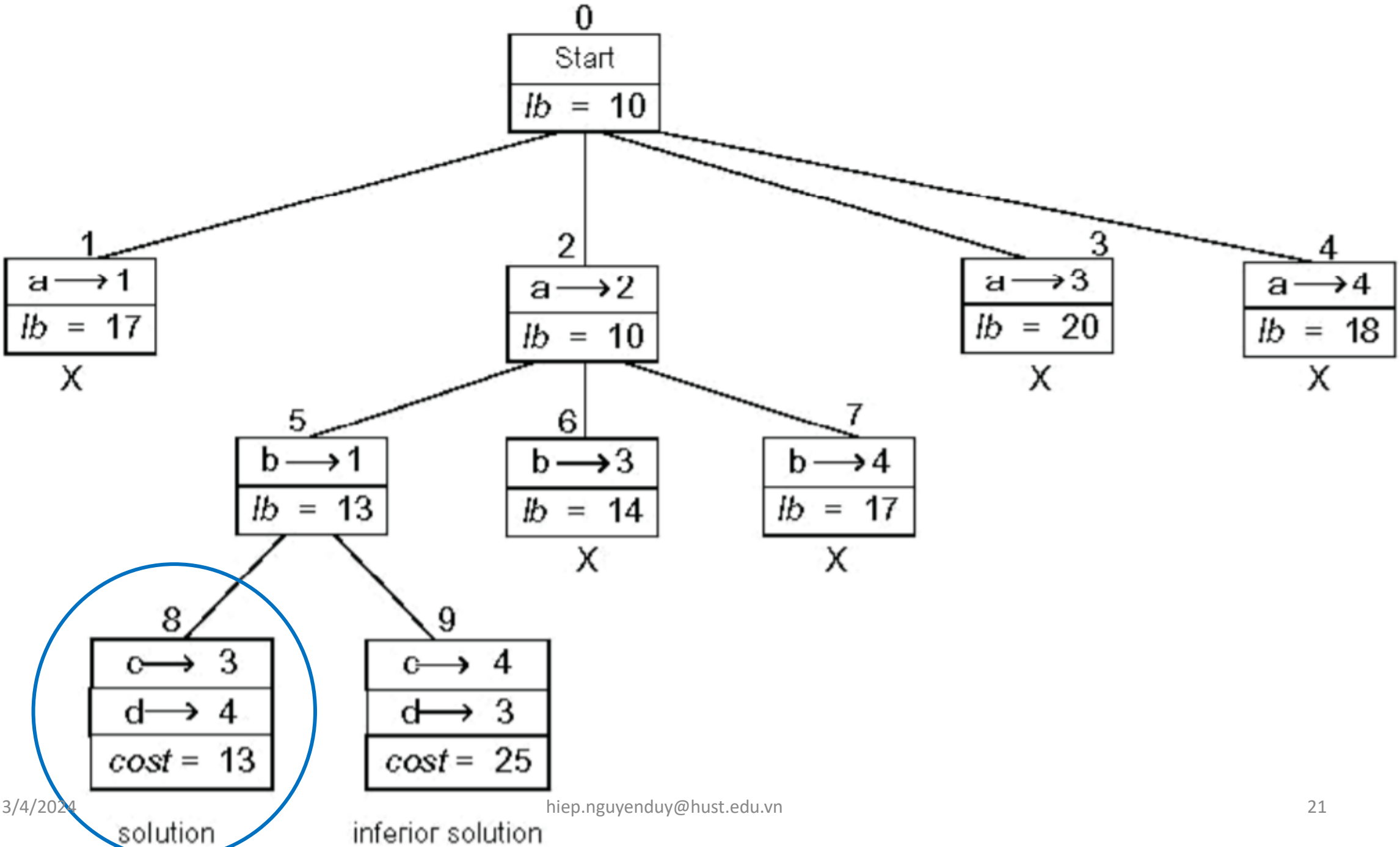
- Chọn gán công việc cho người với chi phí nhỏ nhất trước
- Giới hạn dưới – lower bound sẽ được tính lại tại mỗi nút trên cây trạng thái



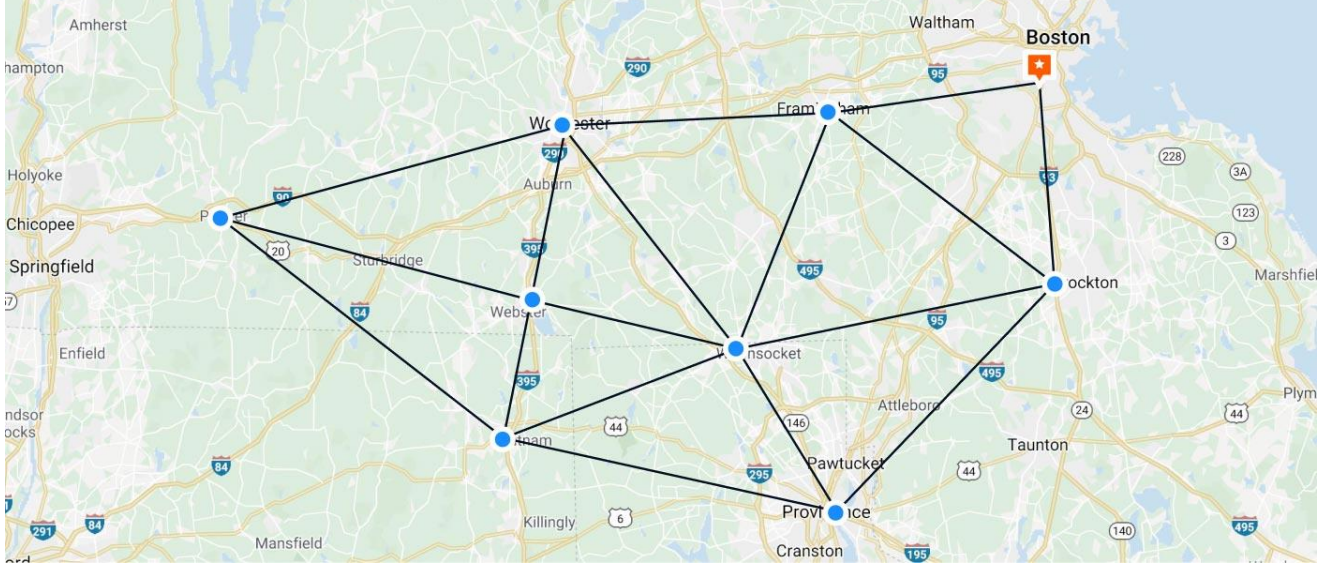
Thuật toán nhánh cận

- Ưu tiên nhánh có Lower Bound nhỏ nhất trước




0
Start
lb = 10
1
a → 1
lb = 17
X
2
a → 2
lb = 10
3
a → 3
lb = 20
X
4
a → 4
lb = 18
X
5
b → 1
lb = 13
6
b → 3
lb = 14
X
7
b → 4
lb = 17
X
8
c → 3
d → 4
cost = 13
solution
9
c → 4
d → 3
cost = 25
inferior solution

Thuật toán nhánh cận



- Ví dụ 2. Bài toán người bán hàng dạo (người du lịch) Traveling Salesperson Problem
  - Có n thành phố 1, 2, …, n.
  - Chi phí đi từ thành phố i đến thành phố j là $c(i, j)$.
  - Hãy tìm một hành trình xuất phát từ thành phố thứ 1, đi qua các thành phố khác, mỗi thành phố đúng 1 lần và quay về thành phố 1 với tổng chi phí nhỏ nhất
- Mô hình hoá
  - Phương án $x = (x_1, x_2, \ldots, x_n)$ trong đó $x_i \in \{1, 2, \ldots, n\}$
  - Ràng buộc C: $x_i \neq x_j$, $\forall\ 1 \leq i < j \leq n$
  - Hàm mục tiêu

$$f(x) = c(x_1, x_2) + c(x_2, x_3) + \ldots + c(x_n, x_1) \rightarrow \min$$

Thuật toán nhánh cận

- Thuật toán vét cạn - duyệt toàn bộ:
  - Liệt kê tất cả các phương án bằng phương pháp đệ quy quay lui
  - Với mỗi phương án, tính toán hàm mục tiêu
  - Giữ lại phương án có hàm mục tiêu nhỏ nhất

$$T(n) = O(n!)$$

```
TRY(k)
  Begin
    Foreach v thuộc A_k
     if check(v,k)
       Begin
         x_k = v;
         if(k = n)
           ghi_nhan_cau_hinh;
           cập nhật kỷ lục f*;
         else TRY(k+1);
       End
  End
Main()
Begin
  TRY(1);
End
```

Thuật toán nhánh cận

- Thuật toán nhánh và cận:
  - Phương án bộ phận (a_1,…, a_k) trong đó a_1 gán cho x_1,… a_k gán cho x_k
  - Phương án (a_1,…, a_k, b_{k+1},…,b_n) là một phương án đầy đủ được phát triển từ (a_1,…,a_k) trong đó b_{k+1} gán cho x_{k+1},…, b_n được gán cho x_n
  - Với mỗi phương án bộ phận (x_1,…, x_k), hàm cận dưới $g(x_1,…, x_k)$ có giá trị không lớn hơn giá trị hàm mục tiêu của phương án đầy đủ phát triển từ (x_1,…,x_k)
  - Nếu $g(x_1,…,x_k) \geq f^*$ thì không phát triển lời giải từ (x_1,…,x_k)

```
TRY(k) {
  Foreach v thuộc A_k
      if check(v,k) {
        x_k = v;
        if(k = n) {
            ghi_nhan_cau_hinh;
            cập nhật kỷ lục f*;
        } {
            if g(x1,…,x_k) < f*
              TRY(k+1);
        }
      }
}
Main()
{
  f* = ∞;
  TRY(1);
}
```

Bài toán người bán hàng dạo

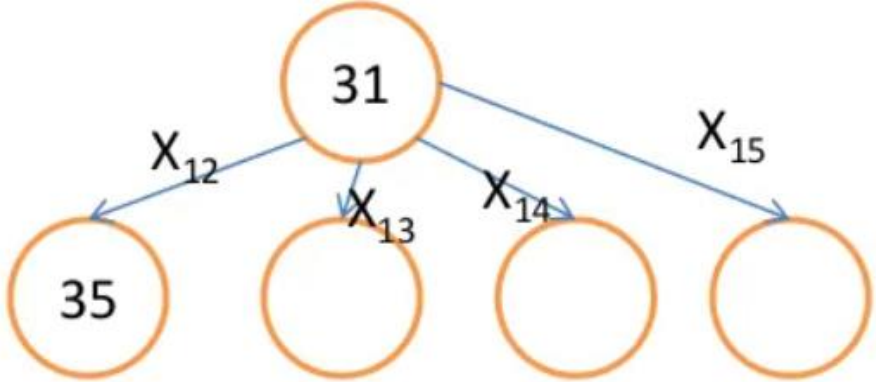
	1	2	3	4	5
1	-	10	8	9	7
2	10	-	10	5	6
3	8	10	-	8	9
4	9	5	8	-	6
5	7	6	9	6	-

- Ma trận chi phí di chuyển là đối xứng
- Thỏa mãn bất đẳng thức tam giác

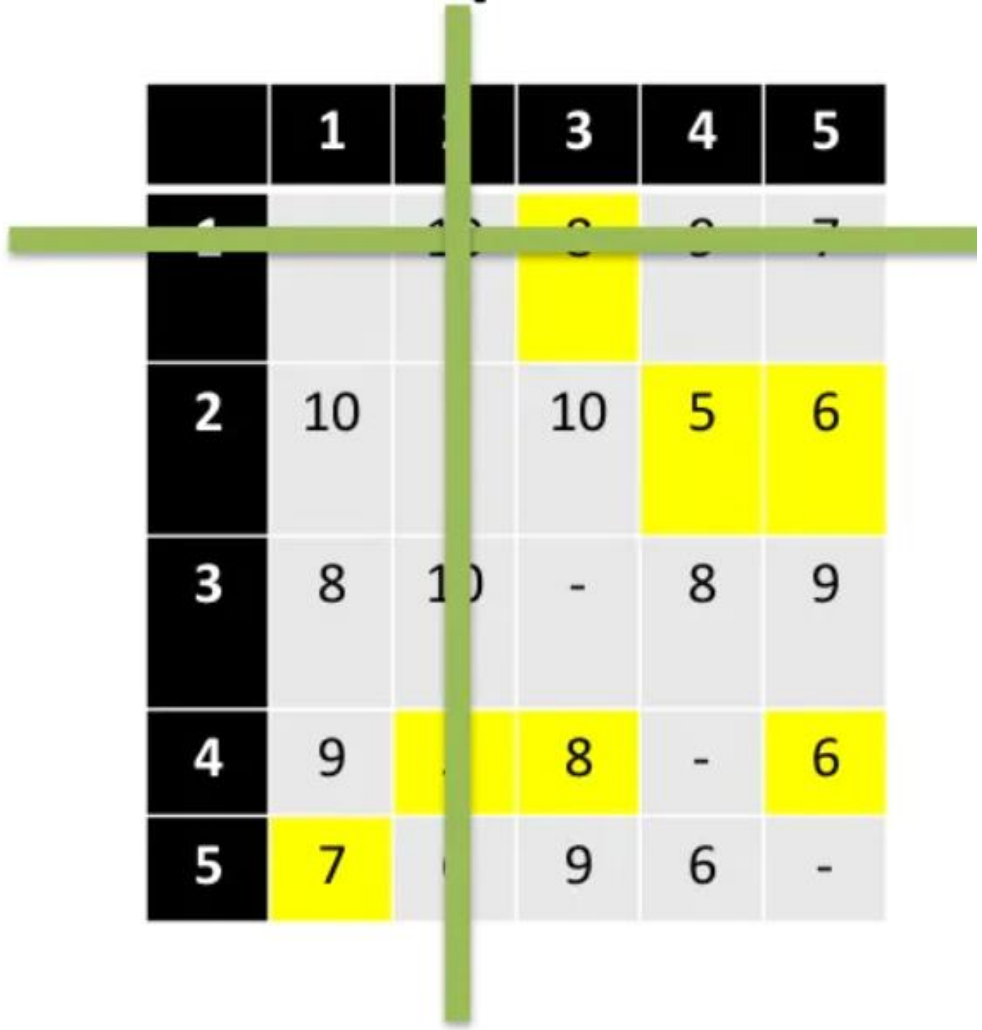
$$d_{ij} + djk >= d_{ik}$$

- Lower bound là chi phí nhỏ nhất tới mỗi thành phố
  - Chi phí thực sẽ phải >= chi phí nhỏ nhất này
  - Trong ví dụ theo cột là 7+5+8+5+6 = 31, và theo hàng là 7+5+8+5+6 = 31
  - Ta sẽ xây dựng thuật toán nhánh cận dựa trên **row minimum branch-and-bound**
- Hàm mục tiêu tại bước i sẽ là giá trị cạnh d_{ij} + lb của các đỉnh còn lại (bỏ qua i và j)
- Nếu chọn x_{ij} thì set x_{ji} là -

https://www.slideshare.net/SaravananNatarajan2/tsp-branch-andbound



	1	3	4	5
2	-	10	5	6
3	8	-	8	9
4	9	8	-	6
5	7	9	6	-

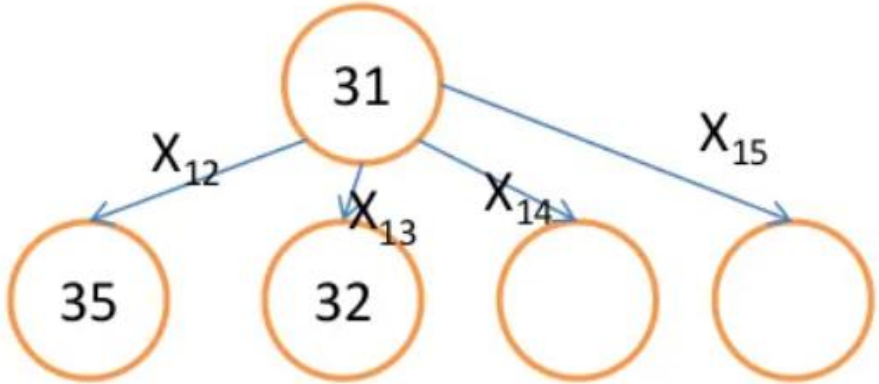


	1		3	4	5
2	10		10	5	6
3	8	1	-	8	9
4	9		8	-	6
5	7		9	6	-

For X_{12}

10+5+8+6+6=35

https://www.slideshare.net/SaravananNatarajan2/tsp-branch-andbound



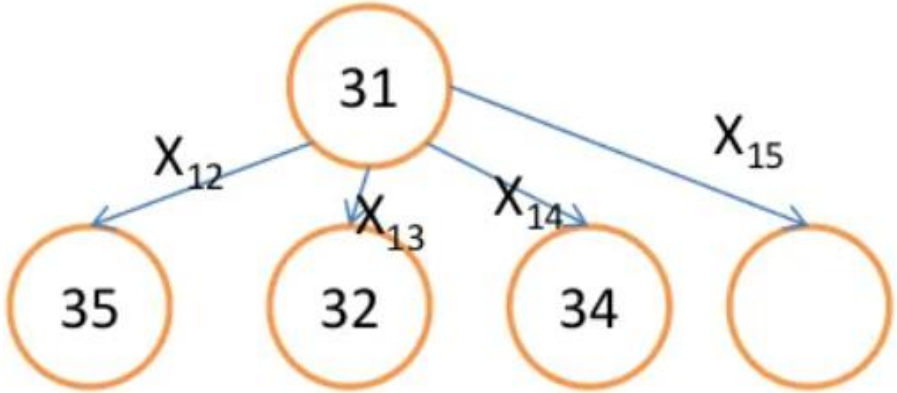
	1	2	4	5
2	10	-	5	6
3	-	10	8	9
4	9	5	-	6
5	7	6	6	-

	1	2		4	5
2	10	-	1	5	6
3	8	10		8	9
4	9	5		-	6
5	7	6		6	-

For X_{13}

8+5+8+5+6=32

https://www.slideshare.net/SaravananNatarajan2/tsp-branch-andbound

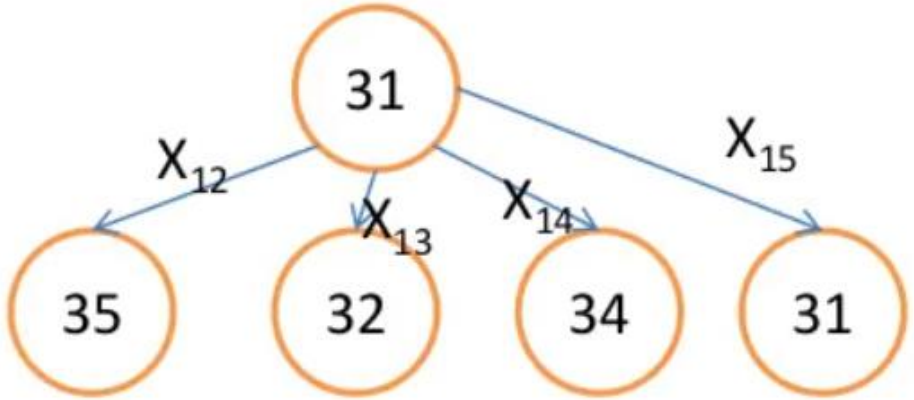


	1	2	3	5
2	10	-	10	6
3	8	10	-	9
4	-	5	8	6
5	7	6	9	-

	1	2	3	4	5
2	10	-	10	5	6
3	8	10	-	8	9
4	9	5	8	-	6
5	7	6	9	6	-

For X_{14}
9+6+8+5+6=34

https://www.slideshare.net/SaravananNatarajan2/tsp-branch-andbound



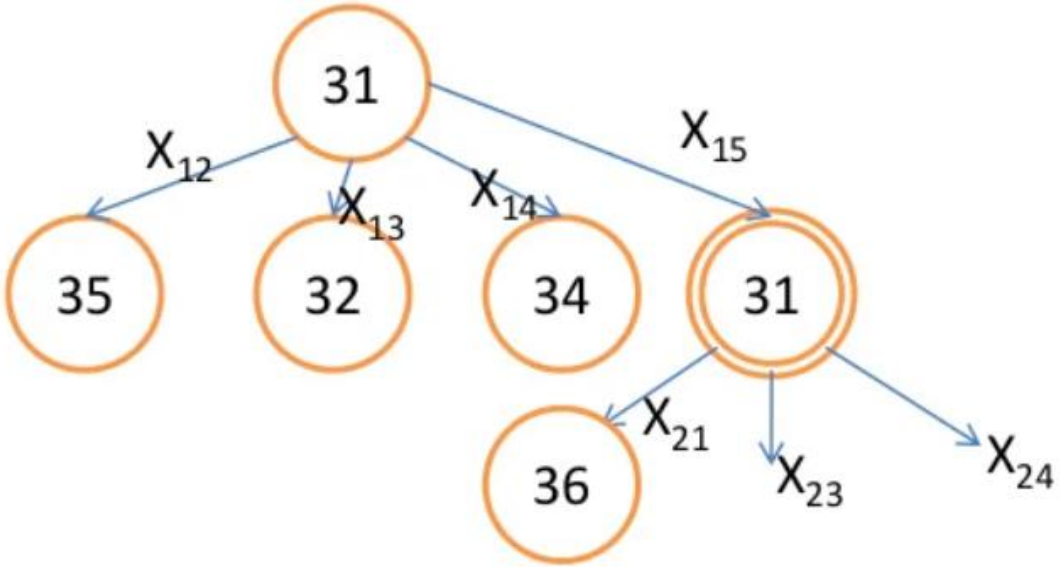
	1	2	3	4
2	10	-	10	5
3	8	10	-	8
4	9	5	8	-
5	-	6	9	6

	1	2	3	4	5
1					
2	10	-	10	5	6
3	8	10	-	8	9
4	9	5	8	-	6
5	7	6	9	6	-

For X_{15}

7+5+8+5+6=31

https://www.slideshare.net/SaravananNatarajan2/tsp-branch-andbound

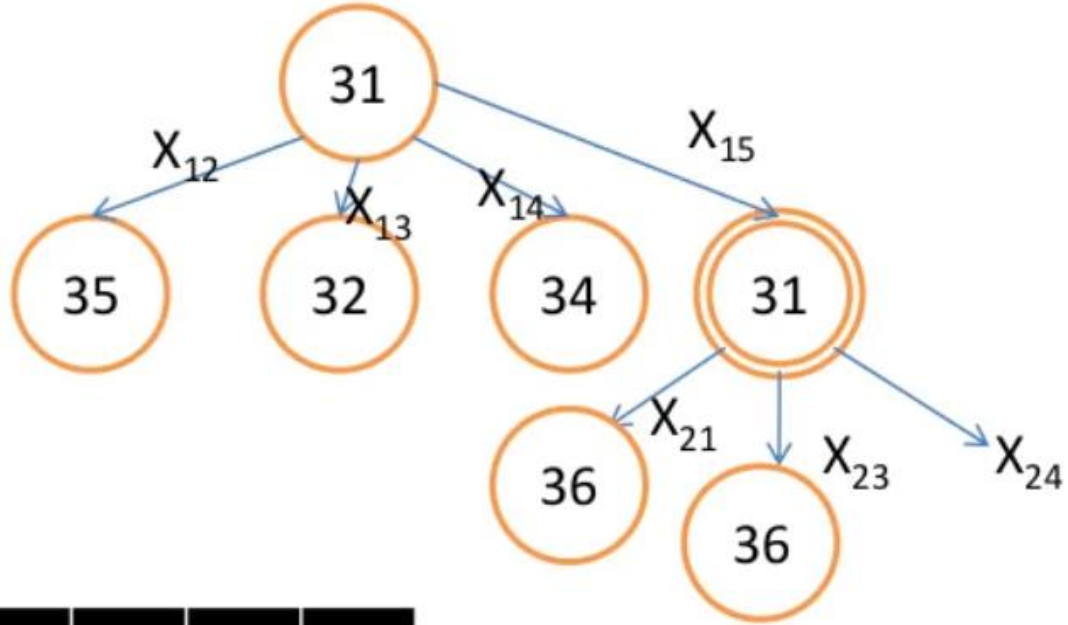


	2	3	4
3	10	-	8
4	5	8	-
5	-	9	6

		2	3	4	
3	8	10	-	8	
4	9	5	8	-	
5	7	6	9	6	

For X_{15}
And X_{21}
7+10+8+5+6=36

https://www.slideshare.net/SaravananNatarajan2/tsp-branch-andbound

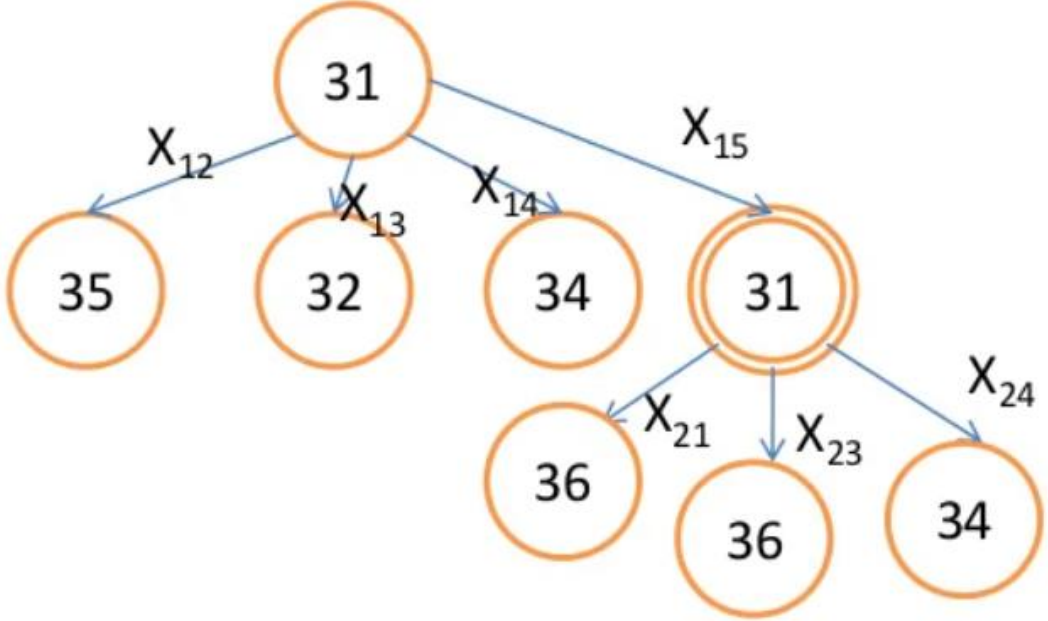


	1	2	4
3	-	10	8
4	9	5	-
5	7	6	6

	1	2	4
3	8	10	8
4	9	5	-
5	7	6	6

For X_{15}
And X_{23}
7+10+8+5+6=36

https://www.slideshare.net/SaravananNatarajan2/tsp-branch-andbound

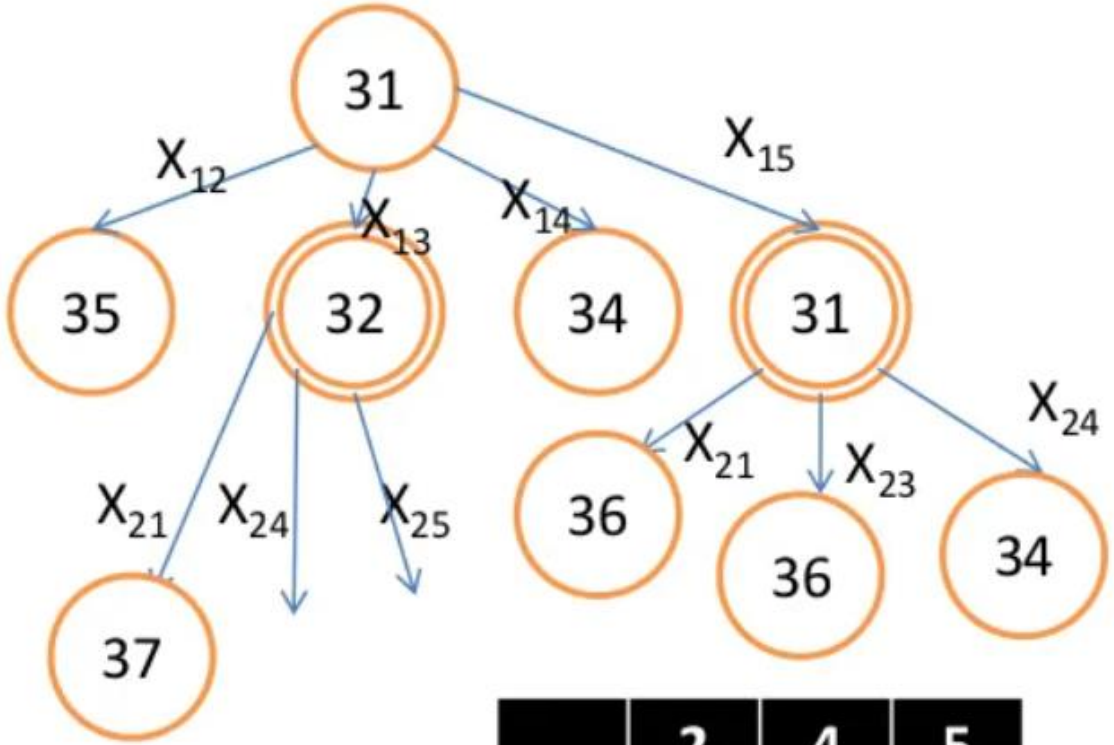


	1	2	3
3	8	10	-
4	9	-	8
5	7	6	9

	1	2	3
3	8	10	-
4	9	5	8
5	7	6	9

For X_{15}
And X_{24}
7+5+8+8+6=34

https://www.slideshare.net/SaravananNatarajan2/tsp-branch-andbound

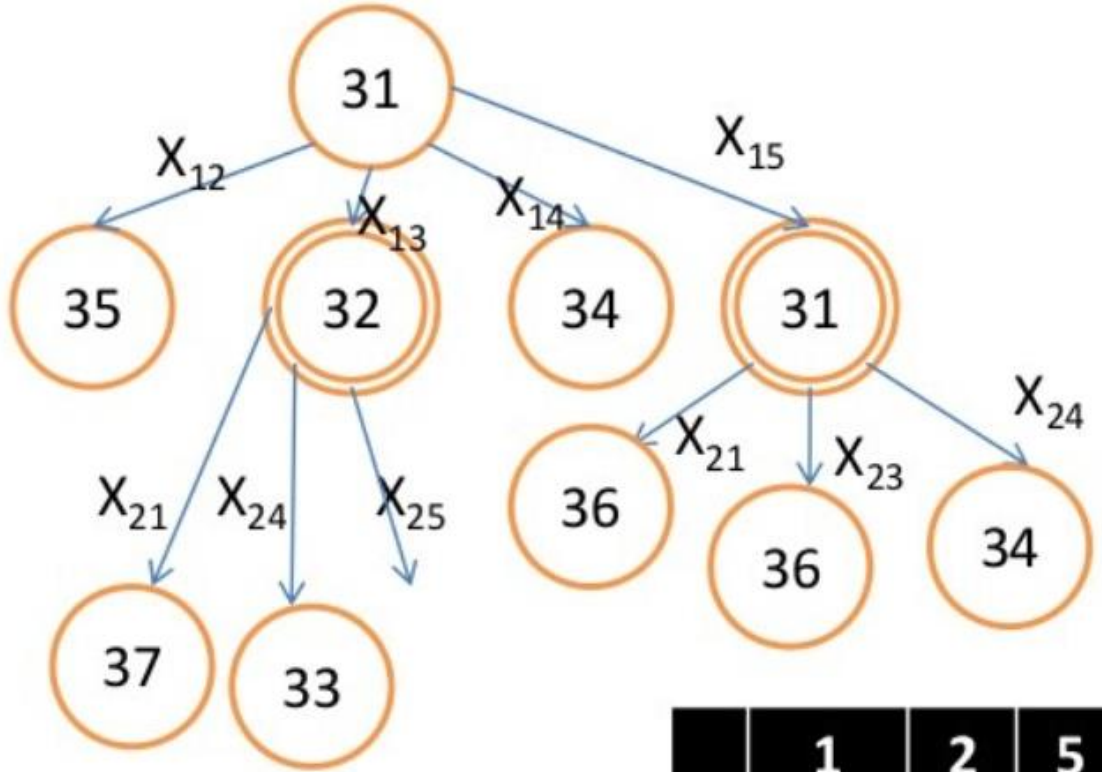


	2	4	5
3	-	8	9
4	5	-	6
5	6	6	-

		2		4	5
1		10		9	7
2					
3		10		8	9
4		5		-	6
5		6		6	-

For X_{13}
And X_{21}
8+10+8+5+6=37

https://www.slideshare.net/SaravananNatarajan2/tsp-branch-andbound

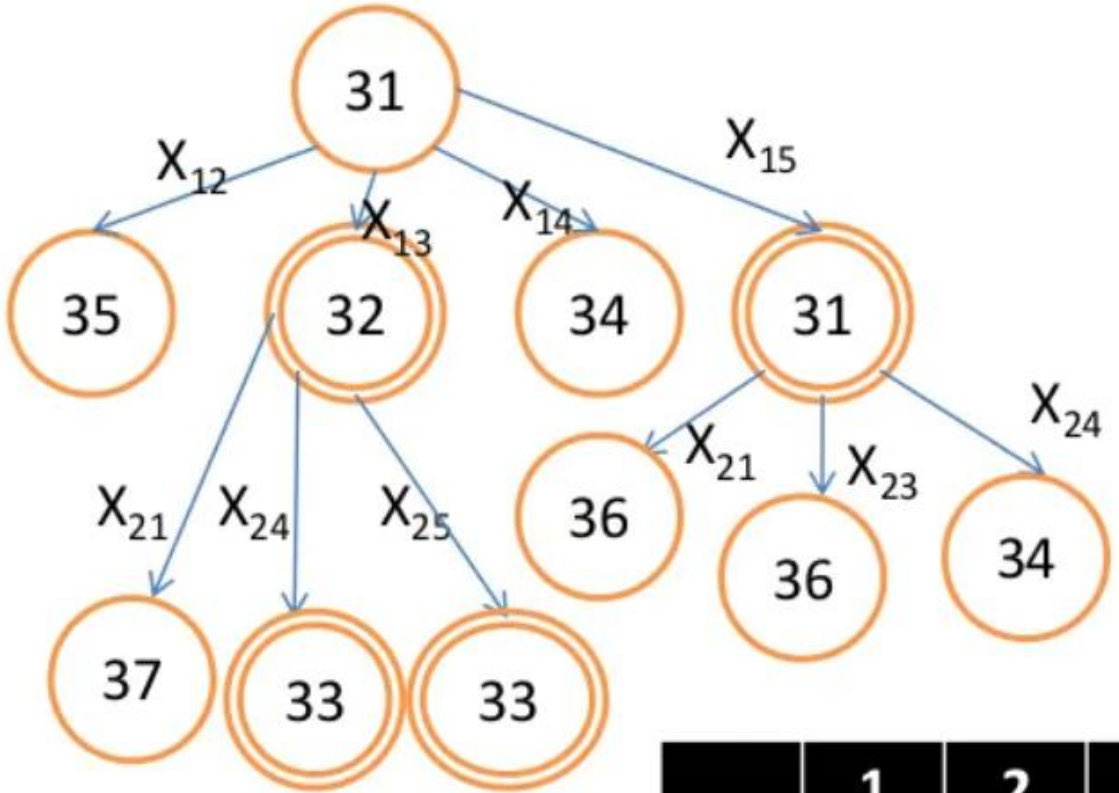


	1	2	5
3	8	10	9
4	9	-	6
5	7	6	-

	1	2			5
1		10			7
2	10	-	1		6
3	8	10			9
4	9	5			6
5	7	6			-

For X_{13}
And X_{24}
8+5+8+6+6=33

https://www.slideshare.net/SaravananNatarajan2/tsp-branch-andbound

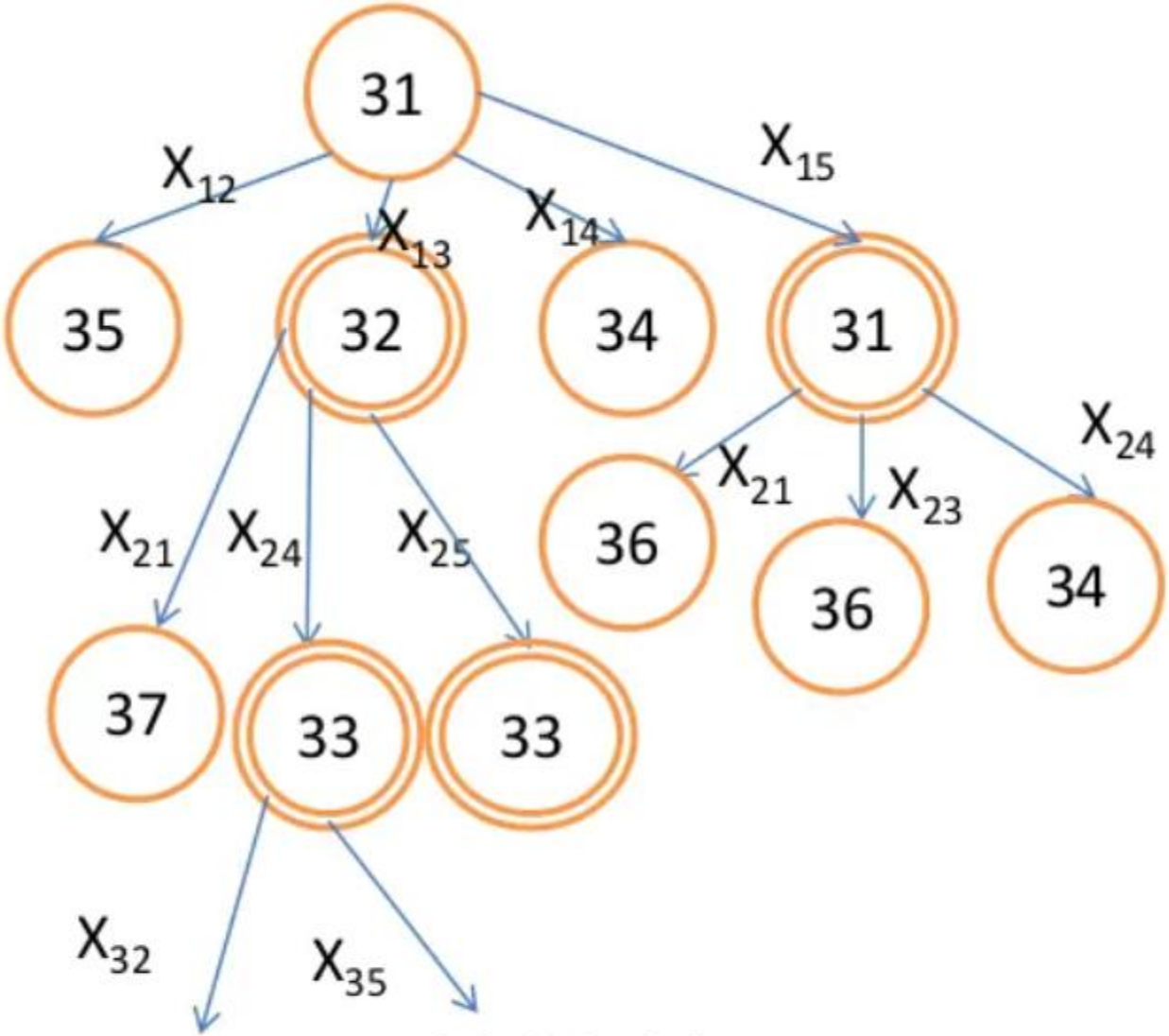


	1	2	4
3	8	10	8
4	9	5	-
5	7	-	6

	1	2	4
3	8	10	8
4	9	5	-
5	7	6	6

For X_{13}
And X_{25}
8+6+8+5+6=33

https://www.slideshare.net/SaravananNatarajan2/tsp-branch-andbound



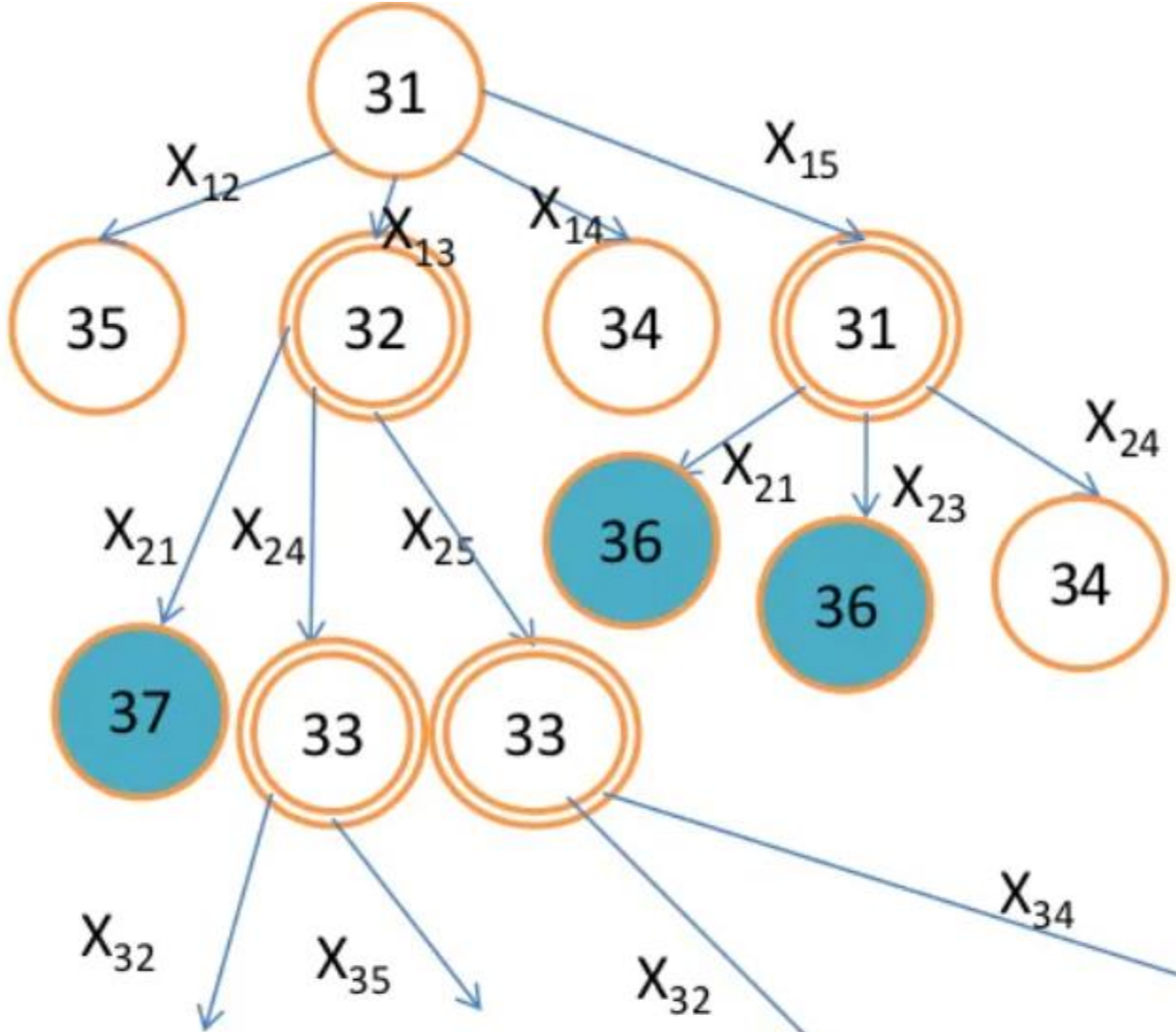
	1	2	3	4	5
1	-	10	8	9	7
2	10	-	10	5	6
3	8	10	-	8	9
4	9	5	8	-	6
5	7	6	9	6	-

1-3-2-4-5-1
Distance=8+10+
5+6+7=36

1-3-5-2-4-1
Distance=8+9+6
+5+9=37

Fill nốt các đỉnh cho đủ hành trình (khi còn thiếu 2 đỉnh)

https://www.slideshare.net/SaravananNatarajan2/tsp-branch-andbound



	1	2	3	4	5
1	-	10	8	9	7
2	10	-	10	5	6
3	8	10	-	8	9
4	9	5	8	-	6
5	7	6	9	6	-

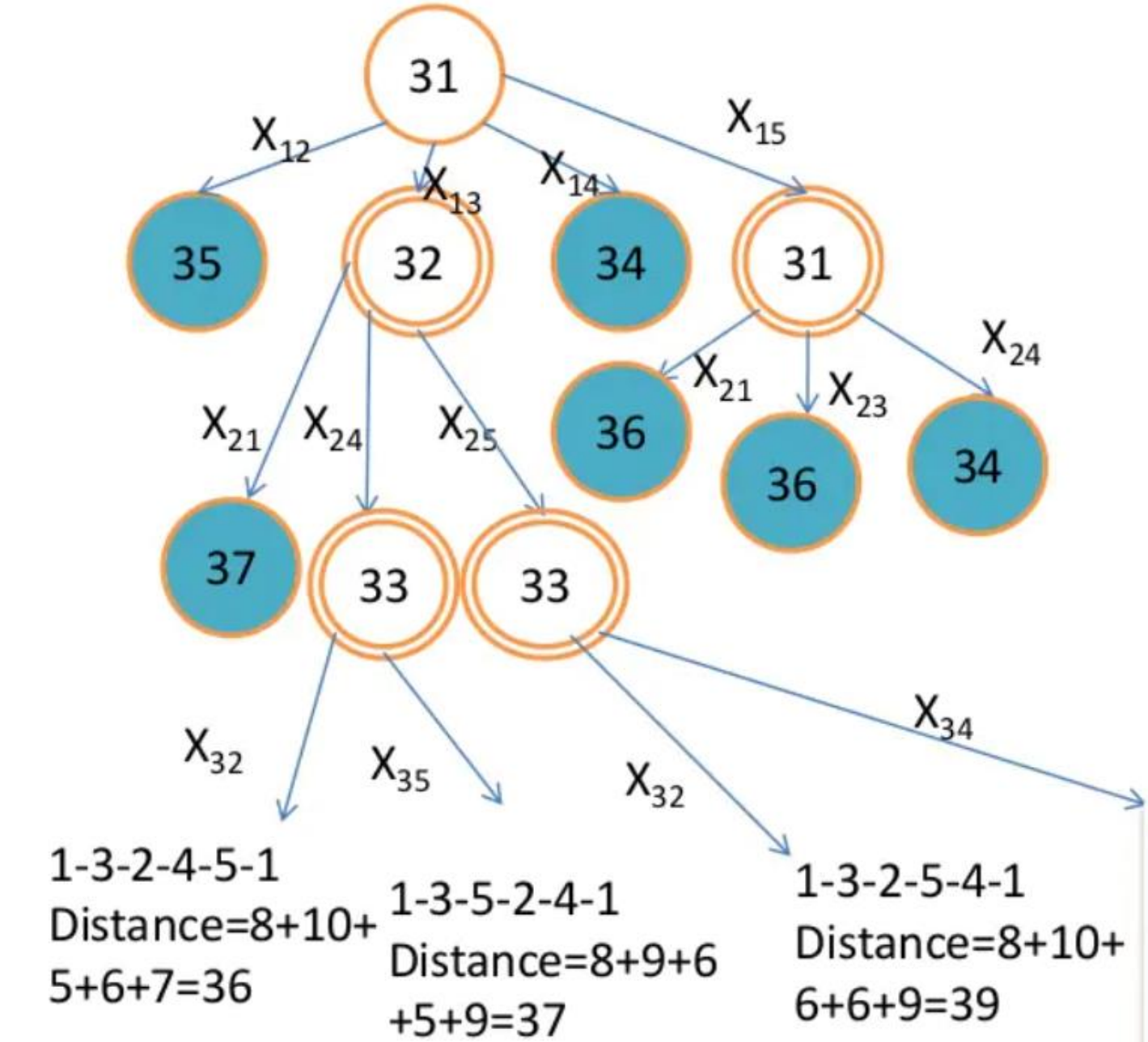
1-3-2-4-5-1
Distance=8+10+5+6+7=36

1-3-5-2-4-1
Distance=8+9+6+5+9=37

1-3-2-5-4-1
Distance=8+10+6+6+9=39

1-3-4-2-5-1
Distance=8+8+5+6+7=34

https://www.slideshare.net/SaravananNatarajan2/tsp-branch-andbound



	1	2	3	4	5
1	-	10	8	9	7
2	10	-	10	5	6
3	8	10	-	8	9
4	9	5	8	-	6
5	7	6	9	6	-

1-3-4-2-5-1
Distance=8+8+5+6+7=34
This is the Optimal Solution.
This is same as
1-5-2-4-3-1

https://www.slideshare.net/SaravananNatarajan2/tsp-branch-andbound

Bài toán người bán hàng dạo - TSP

Các dạng thuật toán nhánh cận khác cho bài toán TSP?

- Travelling salesman problem using Hungarian method
- Travelling salesman branch and bound (penalty) method
- Travelling salesman branch and bound method
- Travelling salesman nearest neighbor method
- Travelling salesman diagonal completion method

https://cbom.atozmath.com/example/CBOM/Assignment.aspx?he=e&q=TSP

- *VD một branch and bound với hàm cận dưới khác*
  - c_m là chi phí nhỏ nhất trong số các chi phí đi giữa 2 thành phố khác nhau
  - Phương án bộ phận $(x_1,\ldots, x_k)$
    - Chi phí bộ phận $f = c(x_1, x_2) + c(x_2, x_3) + \ldots + c(x_{k-1}, x_k)$
    - Hàm cận dưới

$$g(x_1,\ldots, x_k) = f + c_m \times (n-k+1)$$

```
void TRY(int k){
  for(int v = 1; v <= n; v++){
    if(marked[v] == false){
      a[k] = v;
      f = f + c[a[k-1]][a[k]];
      marked[v] = true;
      if(k == n){
        process();
      }else{
        int g = f + cmin*(n-k+1);
        if(g < f_min)
          TRY(k+1);
      }
      marked[v] = false;
      f = f - c[a[k-1]][a[k]];
    }
   }
}
```

```
void process() {
  if(f + c[x[n]][x[1]] < f_min){
    f_min = f + c[x[n]][x[1]];
  }
}
void main() {
  f_min = 9999999999;
  for(int v = 1; v <= n; v++)
    marked[v] = false;
  x[1] = 1; marked[1] = true;
  TRY(2);
}
```

Thuật toán nhánh cận

- Bài tập 1. Bài toán dãy con tăng dần dài nhất
  - Đầu vào: dãy số nguyên $a = a_1, a_2, ..., a_n$ (gồm các phần tử đôi một khác nhau)
  - Đầu ra: tìm dãy con (bằng cách loại bỏ 1 số phần tử) của dãy đã cho, để thu được dãy tăng dần có số lượng phần tử là lớn nhất (gọi là dãy con dài nhất)

Ví dụ: Dãy đầu vào 1, 3 ,6, 4, 5, -4, 10, 25, 3

Một dãy con tăng dần là : 1, 3, 6, 10, 25 với độ dài 5

Dãy con tăng dần lớn nhất 1, 3, 4, 5, 10, 25 với độ dài 6

Thuật toán tham lam – greedy algorithm

- Thuật toán tham lam
  - Xây dựng lời giải từng bước (tương tự back tracking hoặc nhánh cận)
  - Luôn lựa chọn phương án tiếp theo là cái dễ thấy và đem lại lợi ích ngày trước mắt (tham lam)
  - Tại mỗi bước, phương án tốt nhất đựa trên quan sát hiện tại sẽ được lựa chọn
  - Phù hợp với các bài toán mà chỉ có 1 cực trị toàn. Lời giải tối ưu toàn cục tìm bằng cách tìm bước đi tối ưu ở mỗi bước chọn.
  - Không phù hợp với các bài toán tối ưu phức tạp, có nhiều cực trị địa phương

Thuật toán tham lam – greedy algorithm

- Lời giải được biểu diễn bởi tập S
- C biểu diễn các ứng cử viên
- select(C): chọn ra ứng cử viên tiềm năng nhất
- solution(S): trả về true nếu S là lời giải của bài toán
- feasible(S): trả về true nếu S không vi phạm ràng buộc nào của bài toán

```
Greedy() {
  S = {};
  while C ≠ ∅ and
        not solution(S){
    x = select(C);
    C = C \ {x};
    if feasible(S ∪ {x}) {
      S = S ∪ {x};
    }
  }
  return S;
}
```

Thuật toán tham lam – gre

- Ví dụ 1. Bài toán đổi tiền
  - Giả sử bạn có các tờ tiền mệnh giá 1, 2, 5 và 10 đơn vị, và
  - Cần đổi cho số tiền là 18
  - Hãy đưa ra cách đổi tiền sao cho phải sử dùng ít tờ tiền nhất có thể

Thuật toán tham lam: dùng các tờ tiền có mệnh giá cao nhất trước
18 = 10 + 5 + 2 + 1 (tổng 4 tờ)

Nhưng nếu các tờ tiền chỉ có mệnh giá 10, 7 và 1
16 = 10 + 1 + 1 + 1 + 1 + 1 + 1 = 7 tờ
Trong khi phương án tối ưu là 7 + 7 + 1 + 1 = 4 tờ

Thuật toán tham lam – greedy algorithm

Ví dụ 2. Bài toán các đoạn không giao nhau (bài toán lập lịch xem film, bài toán xếp lịch làm việc)

- Đầu vào n công việc trên máy với thời điểm bắt đầu và kết thúc, mỗi công việc cần được hoàn thành đầy đủ (từ thời điểm bắt đầu tới hết thời điểm kết thúc)
- Cần phân công công việc cho 1 công nhân sao cho người này làm được nhiều việc nhất có thể

Ví dụ 3. Bài toán phân số Ai Cập - Egyptian Fraction

- Đầu vào là 1 số dạng a/b (trong đó a<b)
- Tìm cách biểu diễn lại a/b dưới dạng 1/c + 1/d + 1/e +..+1/z với số lượng phân số này là nhỏ nhất
- VD. 2/3 = 1/2 + 1/6, hoặc 12/13 = 1/2 + 1/3 + 1/12 + 1/156

Ý tưởng: Với phân số dạng a/b, tìm phân số dạng 1/c lớn nhất mà nhỏ hơn a/b, sau đó lặp lại với phần dư cho tới khi phần dư là 1. Phân số dạng lớn nhất thì là 1/ [ceil(b/a)]

Thuật toán tham lam – greedy algorithm

- Ví dụ 3. Bài toán dây chuyền sản xuất
  - Đầu vào là danh sách các công việc với thời hạn cần hoàn thành (dead line) và phần thưởng cho việc hoàn thành công việc đó
  - Mỗi công việc chỉ cần làm trong 1 đơn vị thời gian
  - Công việc cần hoàn thành trước hoặc đúng dead line
  - Hãy đưa ra cách chọn công việc để tối đa hóa phần thưởng

JobID	Deadline	Profit
a	4	20
b	1	10
c	1	40
d	1	30

Với danh sách công việc trên thì ta chỉ có thể
chọn 1 trong 3 công việc b,c,d do cùng deadline,
sau đó chọn thêm được công việc a
→ chọn c,a với tổng phần thưởng là 60

https://www.geeksforgeeks.org/job-sequencing-problem/

Bài toán dây chuyền sản xuất

JobID	Deadline	Profit
a	2	100
b	1	19
c	2	27
d	1	25
e	3	15

- Với danh sách 5 công việc trên ta có thể chọn là
  - d, a, e với tổng thưởng là 140
  - Hoặc thực hiện c, a, e với tổng thưởng là 142
- Thuật toán tham lam?
  - TT1: Chọn công việc có phần thưởng cao nhất trước?
    - Sắp xếp các công việc theo chi phí phần thưởng giảm dần
    - Lần lượt theo deadline tăng dần
    - Thêm công việc vào danh sách nếu ko vi phạm dead line
  - TT2: Chọn nhiều công việc có thể làm nhất trước
    - Sắp xếp công việc theo deadline tăng dần
    - Chọn công việc tiếp theo cận deadline nhất và có thưởng cao nhất (sẽ làm được nhiều việc nhất)
  - TT3: Sắp xếp các công việc theo phần thưởng cao nhất trước và deadline giảm dần
    - Duyệt theo deadline giảm dần, gán vào deadline công việc có phần thưởng cao nhất trước

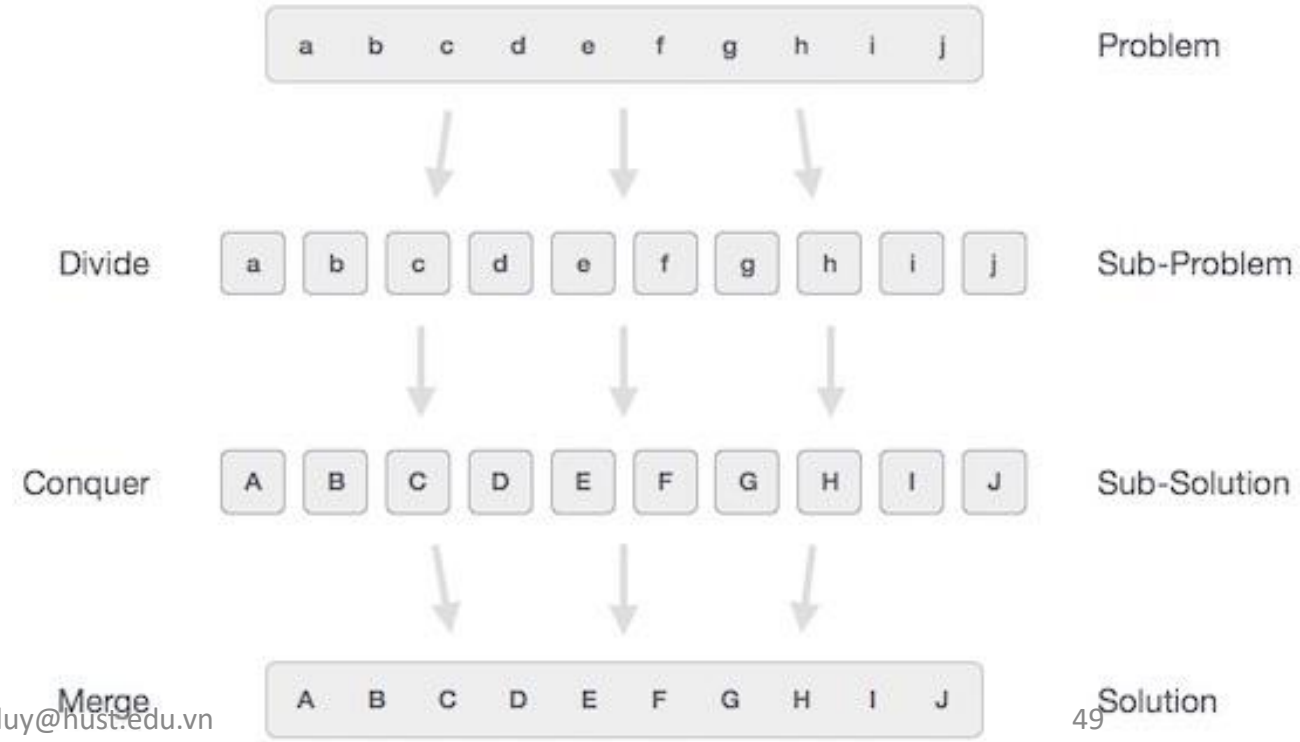
https://www.geeksforgeeks.org/job-sequencing-problem/

Thuật toán tham lam – greedy algorithm

- Ví dụ 4. Bài toán cặp ngoặc cân bằng
  - Bạn có xâu gồm n ngoặc đóng và n ngoặc mở
  - Chỉ được phép hoán đổi 2 ngoặc kề nhau
  - Hãy tìm số phép hoán đổi ít nhất để xâu chứa n cặp ngoặc cân bằng (ngoặc mở rồi mới đến ngoặc đóng)
- VD. xâu []][][cần 2 phép hoán đổi là (3,4) → [][]][, rồi (5,6) → [][][]
  - Xâu [[][]] → không cần hoán đổi vì nó đã cân bằng
- Cách giải?
  - Đổi chỗ ngoặc mở về đầu và ngoặc đóng về cuối ? $O(n^2)$
  - Liệu có cách khác với thời gian O(n)? Chỉ cần đổi chỗ cặp ngoặc cần thiết

Thuật toán chia để trị - divide and conquer

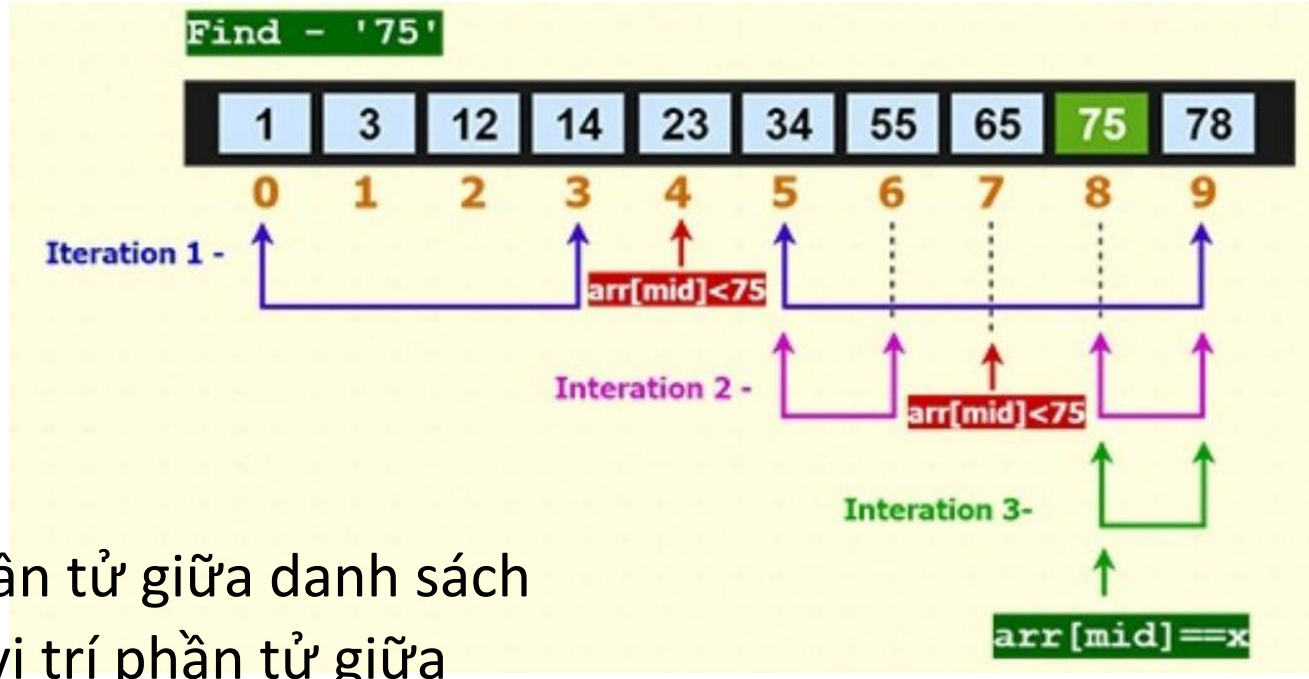
- Thuật toán chia để trị
  - Chia bài toán ban đầu thành các bài toán nhỏ hơn (độc lập nhau) nhưng cùng dạng với bài toán ban đầu
  - Giải để quy các bài toán nhỏ hơn
  - Tổng hợp lời giải để thu được lời giải bài toán cuối cùng
  - Các bài toán con được chia nhỏ tới mức có thể giải được trực tiếp (nếu chưa đủ nhỏ → chia tiếp)



Thuật toán chia để trị - divide and conquer

- Một số thuật toán áp dụng chia để trị
  - Thuật toán tìm kiếm nhị phân
  - Thuật toán sắp xếp trộn merge sort, sắp xếp nhanh Quick sort
  - Thuật toán nhân 2 số nguyên lớn: Karatsuba algorithm
  - Thuật toán nhân 2 ma trận: Strassen's Matrix Multiplication
  - Cặp điểm gần nhất – closest pair
  - Cooley–Tukey Fast Fourier Transform (FFT) algorithm

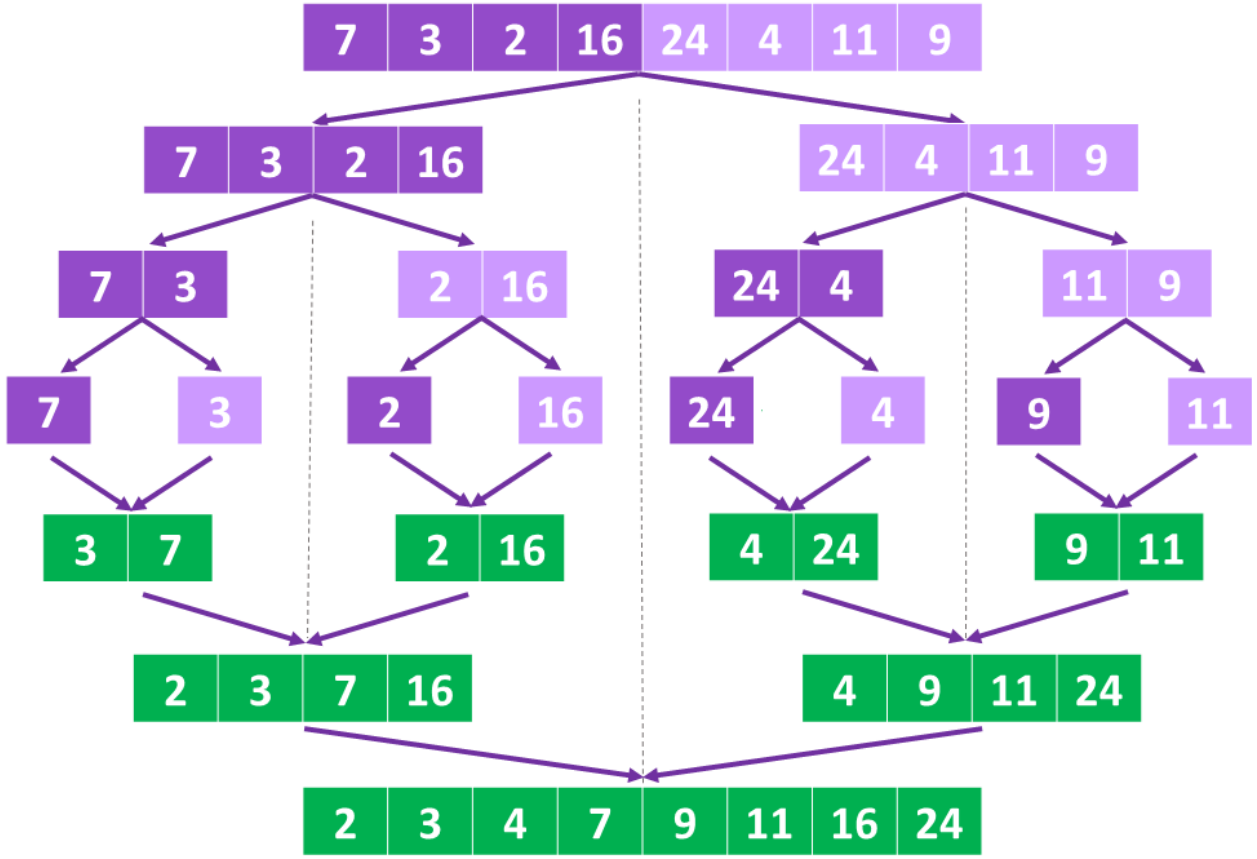
Thuật toán chia để trị



- Thuật toán tìm kiếm nhị phân
  - Dãy ban đầu cần có thứ tự
  - So sánh khóa cần tìm K với khóa của phần tử giữa danh sách
    - Nếu bằng: Tìm thấy, dừng và trả về vị trí phần tử giữa
    - Nếu K> A[mid]: Tìm tiếp tại nửa phải
    - Ngược lại: Tìm kiếm tiếp tại nửa trái
  - Nếu dãy tìm kiếm rỗng → dừng và trả về không tìm thấy
  - Bài toán con: tìm tại nửa trái hoặc nửa phải dãy ban đầu(bài toán con chỉ còn ½ bài toán ban đầu)
  - Bài toán con nhỏ nhất(trường hợp cơ sở): dãy rỗng
  - Tổng hợp lời giải? Lời giải bài toán con chính là lời giải cuối cùng

Thuật toán chia để trị

- Thuật toán sắp xếp trộn – Merge Sort
  - Bước chia: Chia dãy làm đôi nếu số lượng phần tử của dãy hiện tại còn ≥ 2
    Dãy có 1 phần tử là dãy đã sắp xếp (trường hợp cơ sở)
  - Bước trộn: Trộn các dãy đã sắp xếp có số lượng phần tử nhỏ để thu được dãy có phần tử lớn hơn

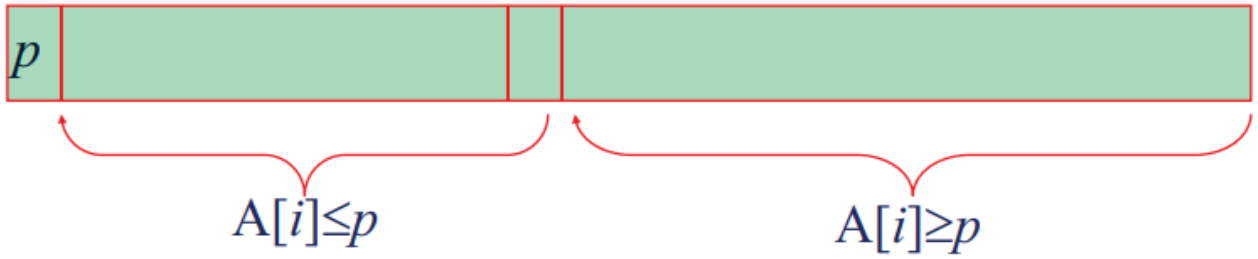



```
void mergeSort(int array[], int const begin, int const end)
{
    if (begin >= end)
        return;

    auto mid = begin + (end - begin) / 2;
    mergeSort(array, begin, mid);
    mergeSort(array, mid + 1, end);
    merge(array, begin, mid, end);
}
```

- Thuật toán sắp xếp trộn – Merge Sort
  - Áp dụng trên mảng không hiệu quả như trên danh sách liên kết vì phải dịch phần tử
  - Muốn tránh dịch phần tử phải dùng mảng phụ

```
// Merges two subarrays of array[].
// First subarray is arr[begin..mid]
// Second subarray is arr[mid+1..end]
void merge(int array[], int const left, int const mid, int const right)
{
    auto const subArrayOne = mid - left + 1;
    auto const subArrayTwo = right - mid;

    // Create temp arrays
    auto *leftArray = new int[subArrayOne], *rightArray = new int[subArrayTwo];

    // Copy data to temp arrays leftArray[] and rightArray[]
    for (auto i = 0; i < subArrayOne; i++)
        leftArray[i] = array[left + i];
    for (auto j = 0; j < subArrayTwo; j++)
        rightArray[j] = array[mid + 1 + j];

    auto indexOfSubArrayOne = 0, // Initial index of first sub-array
        indexOfSubArrayTwo = 0; // Initial index of second sub-array
    int indexOfMergedArray = left; // Initial index of merged array

    // Merge the temp arrays back into array[left..right]
    while (indexOfSubArrayOne < subArrayOne && indexOfSubArrayTwo < subArrayTwo) {
        if (leftArray[indexOfSubArrayOne] <= rightArray[indexOfSubArrayTwo]) {
            array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
            indexOfSubArrayOne++;
        }
        else {
            array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];
            indexOfSubArrayTwo++;
        }
        indexOfMergedArray++;
    }
    // Copy the remaining elements of
    // left[], if there are any
    while (indexOfSubArrayOne < subArrayOne) {
        array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
        indexOfSubArrayOne++;
        indexOfMergedArray++;
    }
    // Copy the remaining elements of
    // right[], if there are any
    while (indexOfSubArrayTwo < subArrayTwo) {
        array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];
        indexOfSubArrayTwo++;
        indexOfMergedArray++;
    }
}
```

Thuật toán chia để trị



- Thuật toán sắp xếp nhanh – Quick sort
  - Chia đôi dãy dựa trên phần tử chốt (VD. phần tử ở đầu dãy, giữa dãy, random,..)
  - Sắp xếp lại dãy dựa trên chốt:
    - Nửa trái là s các phần tử <= chốt, nửa phải sẽ là n-s các phần tử >= chốt
    - Sau đó đổi chỗ chốt và phần tử cuối nửa trái (chốt đã về đúng vị trí của nó)
  - Gọi đệ quy sắp xếp nửa trái và nửa phải (bỏ qua chốt)
- Phân tích
  - Tốt nhất: chốt chia đôi dãy $O(n \log n)$
  - Tồi nhất: Chia lệch $O(n^2)$
  - Trung bình: $O(n \log n)$
- Cải tiến:
  - Chọn chốt là trung vị của 3 phần tử,
  - Dùng thuật toán sắp xếp chèn để sắp xếp khi dãy có ít phần tử (VD. n<10)
  - Cài đặt dùng vòng lặp

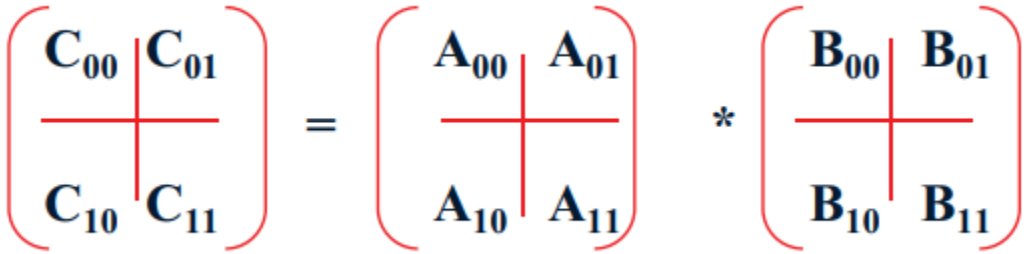
Thuật toán chia để trị

- Thuật toán nhân 2 số nguyên lớn: Karatsuba algorithm
  - Hai số nguyên A và B có thể tới hàng nghìn chữ số (giả sử coi độ dài là n chữ số)
  - Nhân bình thường sẽ có thời gian $O(n^2)$
  - Nhân nhanh hơn?
    - Biểu diễn lại số ban đầu $A = A_1*10^{n/2} + A_2$, $B = B_1*10^{n/2} + B_2$ bằng các số n/2 chữ số
    - $AxB = (A_1*10^{n/2} + A_2)x(B_1*10^{n/2} + B_2)$, việc nhân với 10 chỉ là dịch chữ số nên thời gian nhân quyết định bởi các tích $A_1B_1 + A_1B_2 + A_2B_1 + A_2B_2$.
    - Tích 4 số này vẫn có chi phí là $4 \times \frac{n}{2} \times \frac{n}{2} = O(n^2)$
  - Giải pháp?
    - $C = (A_1-A_2)x(B_1-B_2)=A_1B_1 + A_2B_2 – (A_1B_2+A_2B_1)$ với thời gian tính C cỡ $n^2/4$
    - $A_1B_1 + A_1B_2 + A_2B_1 + A_2B_2 = 2(A_1B_1 + A_2B_2) – C$ với thời gian còn là $3n^2/4$

$$T(n) = 3T(n/2) + O(n), \text{ cỡ } O(n^{1.59})$$

Thuật toán chia để trị

- Thuật toán nhân ma trận: Strassen's matrix multiplication



- Nhân 2 ma trận nxn thời gian là $O(n^3)$, cải tiến?

- Chú ý nếu ma trận ko phải nxn ta có thể thêm các số 0 để nó thành ma trận vuông cỡ nxn

Strassen's matrix multiplication

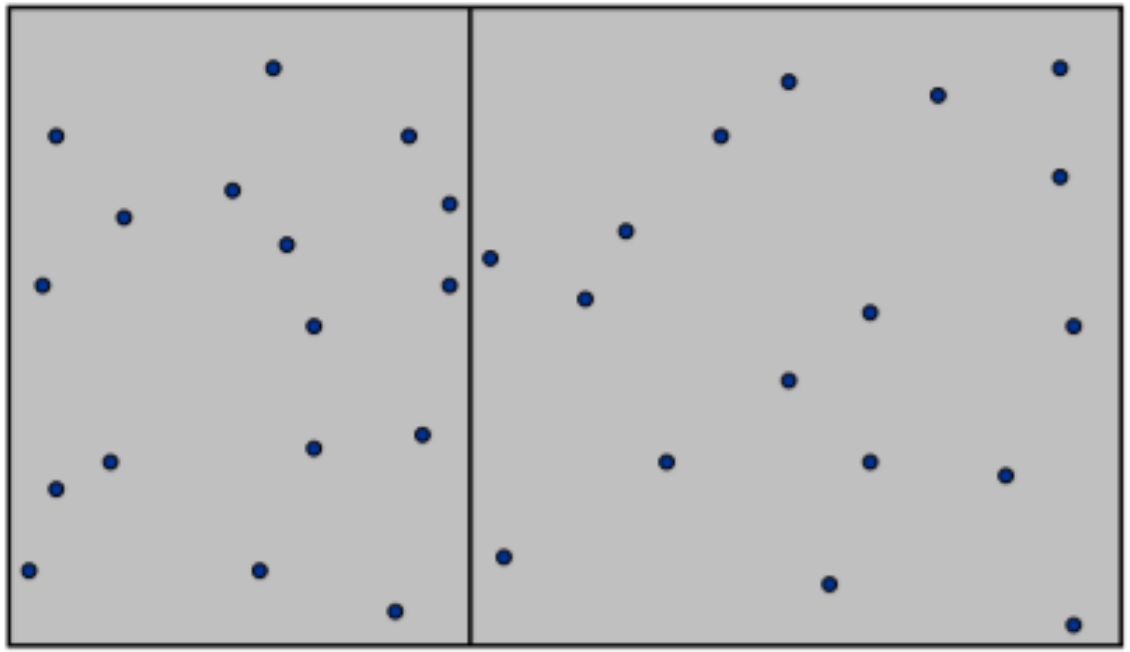
$$\begin{pmatrix} C_{00} & C_{01} \\ C_{10} & C_{11} \end{pmatrix} = \begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} * \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix}$$

$$= \begin{pmatrix} M_1 + M_4 - M_5 + M_7 & M_3 + M_5 \\ M_2 + M_4 & M_1 + M_3 - M_2 + M_6 \end{pmatrix}$$

- $M_1 = (A_{00} + A_{11}) * (B_{00} + B_{11})$
- $M_2 = (A_{10} + A_{11}) * B_{00}$
- $M_3 = A_{00} * (B_{01} - B_{11})$
- $M_4 = A_{11} * (B_{10} - B_{00})$
- $M_5 = (A_{00} + A_{01}) * B_{11}$
- $M_6 = (A_{10} - A_{00}) * (B_{00} + B_{01})$
- $M_7 = (A_{01} - A_{11}) * (B_{10} + B_{11})$

- $T(n) = 7T\left(\frac{n}{2}\right) + O(n^2) = O(n^{2.8074})$

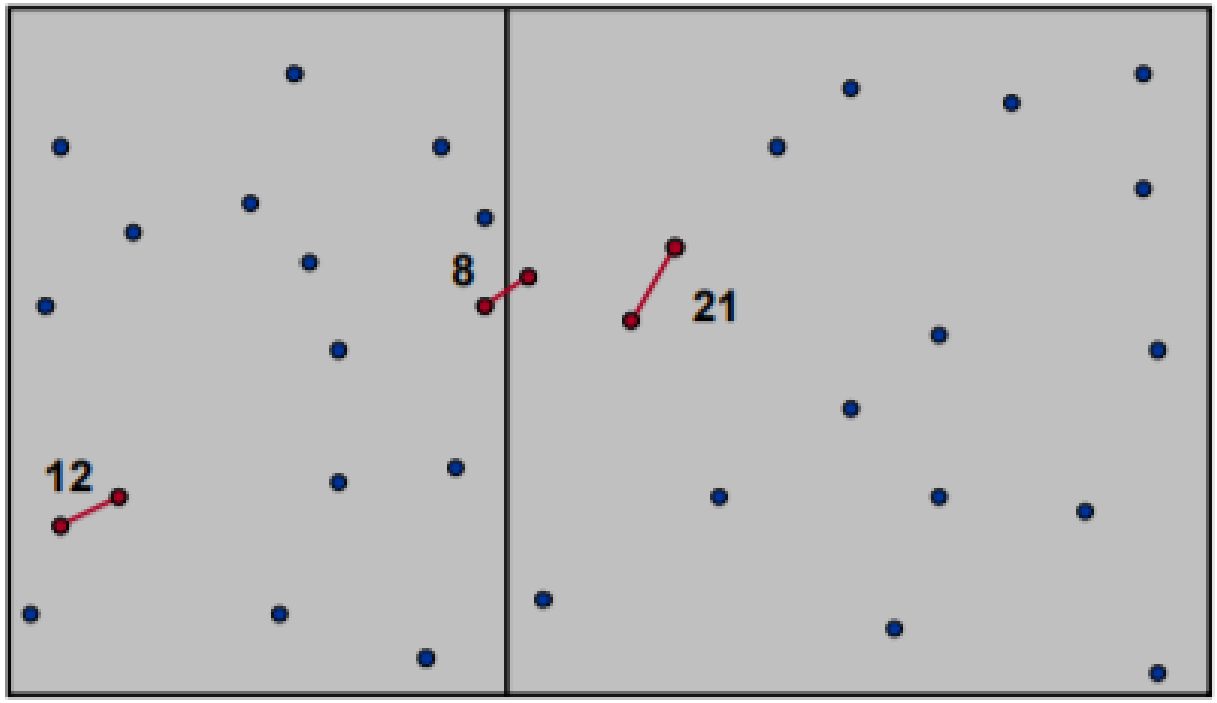
Thuật toán chia để trị



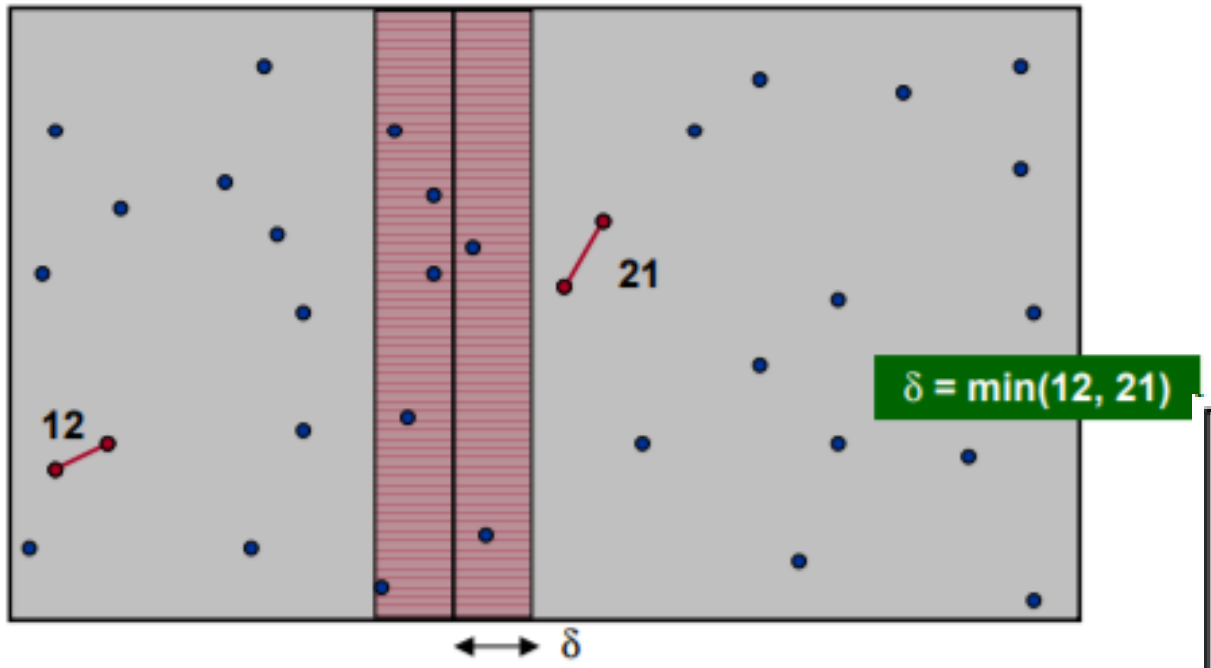
- Bài toán tìm cặp điểm gần nhau trong không gian 2D
  - N điểm trong không gian Euclidean
  - Tìm cặp điểm gần nhau nhất
- Thuật toán vét cạn: $O(n^2)$
- Chia để trị
  - Chia đôi n điểm theo trục x (tìm trung vị) – cần sắp xếp trước? O(nlogn) cho sắp xếp

Bài toán tìm cặp điểm gần nhau

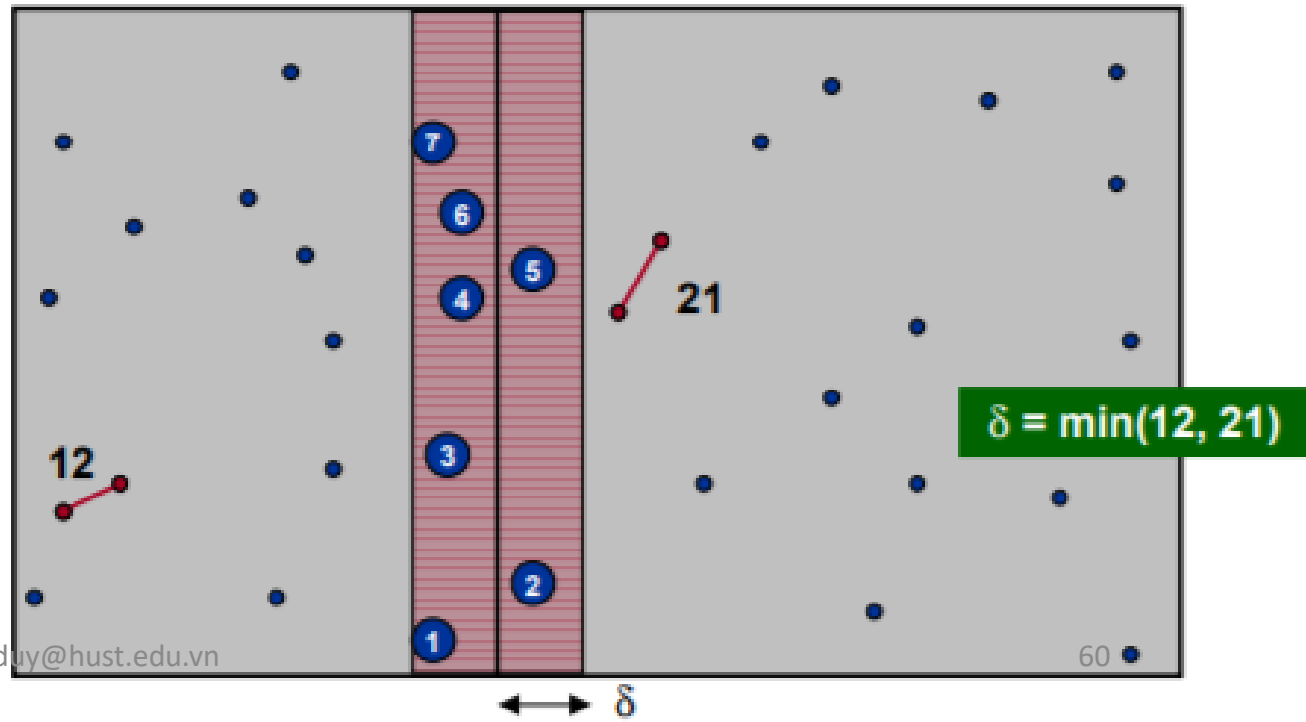
- Cặp điểm gần nhất
  - Hoặc nằm trọn trong nửa trái, hoặc nửa phải → gọi đệ quy
  - Hoặc 1 điểm nằm nửa trái và 1 điểm nằm trong nửa phải
  - Trả về min (của 3 cặp điểm đó)
- Làm thế nào tìm cặp gần nhau ở giữa hiệu quả nhất?
  - Xét tất cả các cặp điểm có thể → không hiệu quả
  - Chỉ xét 1 số cặp điểm tiềm năng



Bài toán tìm cặp điểm gần nhau

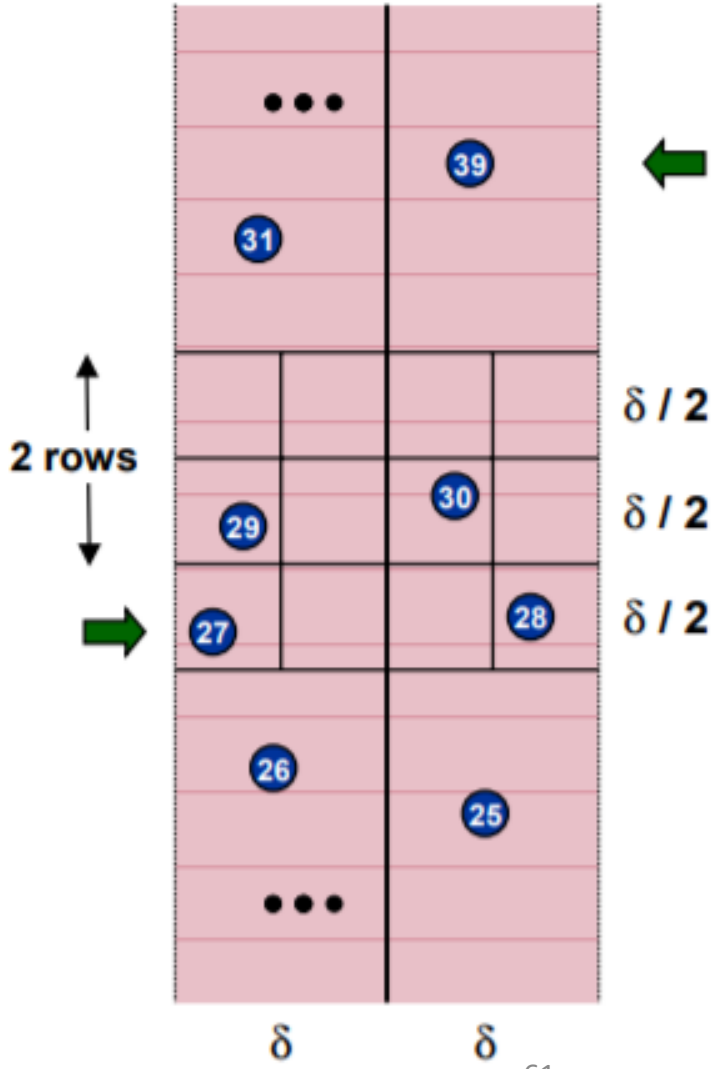


- Tìm nhanh hơn?
  - Chỉ cần xét các điểm trong khoảng cách δ



Bài toán tìm cặp điểm gần nhau

- Tìm nhanh hơn
  - Sắp xếp các điểm theo trục Y,
  - Chỉ xét các điểm tiềm năng trong khoảng cách $\delta/_2$
- Chú ý
  - Sắp xếp O (nlogn)
  - Với thời gian O(n) - > sắp xếp trước
- T(n) = 2T(n/2) + O(nlogn)
- T(n) = 2T(n/2) + O(n) → sắp xếp trước



Thuật toán chia để trị

- Ví dụ 1. Cho dãy gồm n phần tử số nguyên, hãy tìm và trả về phần tử lớn nhất thứ k trong dãy
  Dãy 1, 5, 3, 7, 4, 4, 16, 2 thì k =2 sẽ trả về 7, k=3 trả về 5
  - Sắp xếp dãy : O(nlogn)
  - Chạy k lần tìm giá trị lớn nhất : O(nk)
  - Chia để trị O(n)
- Ví dụ 2. Cho ma trận với các phần tử trên hàng và cột đã có thứ tự và không trùng nhau. Hãy xây dựng hàm tìm kiếm 1 khóa có trong ma trận.

[1 14 25 35]
[2 16 28 38]
[5 20 32 40]
[16 22 36 41]
Tìm 38.

Thuật toán quy hoạch động

- Thuật toán quy hoạch động – Dynamic programming
  - Có cách tiếp cận tương tự chia để trị, chia nhỏ bài toán lớn thành các bài toán con
  - Các bài toán con thường không độc lập (có thể overlap lẫn nhau)
  - Lời giải các bài toán con được tổng hợp lại để thu được lời giải các bài toán lớn hơn (hoặc các bài toán overlap với bài toán con đó)
  - Lời giải các bài toán con thường được lưu vào bảng tra để tránh phải giải lại nhiều lần
- Có 2 cách tiếp cận
  - Top down với bảng tra
  - Bottom up với Tabulation

https://www.geeksforgeeks.org/dynamic-programming

Top-down với bảng tra - Memoization

Fib(n) = Fib(n-1) + Fib(n-2), với n > 1

```
public int CalculateFibonacci(int n) {
   int memoize[] = new int[n+1];
   return CalculateFibonacciRecursive(memoize, n);
 }

 public int CalculateFibonacciRecursive(int[] memoize, int n) {
   if(n < 2)    return n;

   // nếu bài toán đã có lời giải trong bảng tra → trả về kết quả
   if(memoize[n] != 0)      return memoize[n];

   memoize[n] = CalculateFibonacciRecursive(memoize, n-1) + CalculateFibonacciRecursive(memoize, n-2);
   return memoize[n];
 }
```

Bottom-up với Tabulation

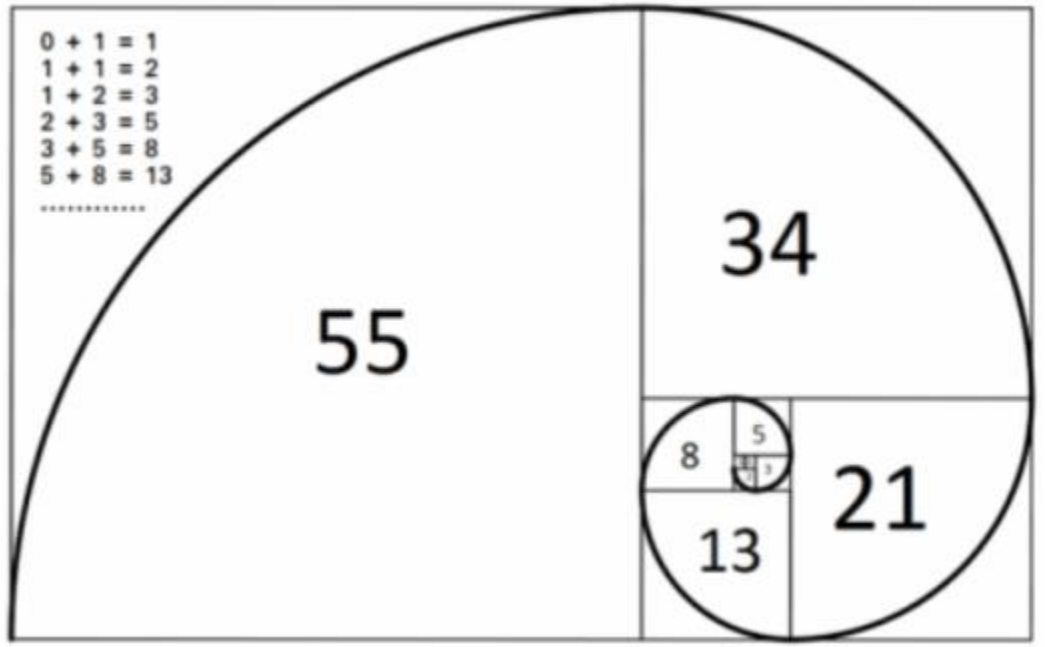
- Giải bài toán theo chiều từ dưới lên (giải các bài toán nhỏ trước)
- Lời giải cũng được lưu trong bảng

Fib(n) = Fib(n-1) + Fib(n-2), với n > 1

```
public int CalculateFibonacci(int n) {
   if (n==0) return 0;
   int dp[] = new int[n+1];

   //base cases
   dp[0] = 0;
   dp[1] = 1;

   for(int i=2; i<=n; i++)
     dp[i] = dp[i-1] + dp[i-2];

   return dp[n];
  }
```

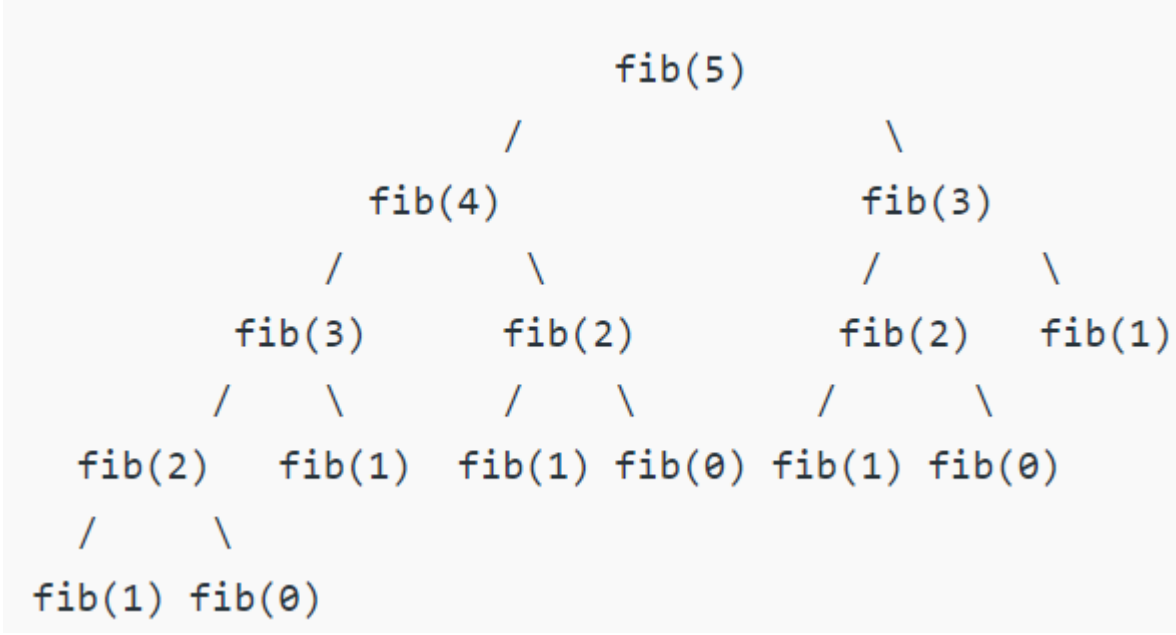
	Tabulation	Memoization
State	State Transition relation is difficult to think	State transition relation is easy to think
Code	Code gets complicated when lot of conditions are required	Code is easy and less complicated
Speed	Fast, as we directly access previous states from the table	Slow due to lot of recursive calls and return statements
Subproblem solving	If all subproblems must be solved at least once, a bottom-up dynamic-programming algorithm usually outperforms a top-down memoized algorithm by a constant factor	If some subproblems in the subproblem space need not be solved at all, the memoized solution has the advantage of solving only those subproblems that are definitely required
Table Entries	In Tabulated version, starting from the first entry, all entries are filled one by one	Unlike the Tabulated version, all entries of the lookup table are not necessarily filled in Memoized version. The table is filled on demand.

Thuật toán quy hoạch động



- Bài toán 1. Tìm giá trị số Fibonacci thứ n
  0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……..

```
int fib(int n)
{
    if (n <= 1)
        return n;
    return fib(n-1) + fib(n-2);
}
```



T(n)=T(n-1)+T(n-2)
Cài đặt tồi, tốn bộ nhớ O(n)
Thời gian cỡ hàm mũ

https://www.geeksforgeeks.org/program-for-nth-fibonacci-number/

Tìm giá trị số Fibonacci thứ n

- Dùng quy hoạch động - Bottom up
  - Thời gian O(n), bộ nhớ O(n)
- Cải thiện bộ nhớ?
  - Không cần dùng tới 1 mảng, chỉ cần 2 biến

```
int fib(int n)
{
    int a = 0, b = 1, c, i;
    if( n == 0)
        return a;
    for(i = 2; i <= n; i++)
    {
       c = a + b;
       a = b;
       b = c;
    }
    return b;
}
```

```
int fib(int n)
{

    int f[n + 2];
    int i;


    f[0] = 0;
    f[1] = 1;


    for(i = 2; i <= n; i++)
    {

       f[i] = f[i - 1] + f[i - 2];
    }
    return f[n];
    }
};
```

Tìm giá trị số Fibonacci thứ n

- Nhanh hơn? O(logn) ?

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}$$

```
void multiply(int F[2][2], int M[2][2]);
void power(int F[2][2], int n);

// Function that returns nth Fibonacci number
int fib(int n)
{
    int F[2][2] = {{1, 1}, {1, 0}};
    if (n == 0)
        return 0;
    power(F, n - 1);

    return F[0][0];
}
```

```
// Optimized version of power() in method 4
void power(int F[2][2], int n)
{
    if(n == 0 || n == 1)
       return;
    int M[2][2] = {{1, 1}, {1, 0}};

    power(F, n / 2);
    multiply(F, F);

    if (n % 2 != 0)
        multiply(F, M);
}


void multiply(int F[2][2], int M[2][2])
{
    int x = F[0][0] * M[0][0] + F[0][1] * M[1][0];
    int y = F[0][0] * M[0][1] + F[0][1] * M[1][1];
    int z = F[1][0] * M[0][0] + F[1][1] * M[1][0];
    int w = F[1][0] * M[0][1] + F[1][1] * M[1][1];

    F[0][0] = x;
    F[0][1] = y;
    F[1][0] = z;
    F[1][1] = w;
}
```

Thuật toán quy hoạch động

$$Bell(n) = \sum_{k=0}^{n} S(n,k)$$

- Ví dụ 2. Tìm số cách chia nhóm các phần tử trong 1 tập hợp n phần tử – Bell Number
  - N=2 → 2 cách { {1}, {2} } và {{1, 2} }
  - N=3 → 5 cách { {1}, {2}, {3} }, { {1}, {2, 3} }, { {2}, {1, 3} }, {3}, {1, 2} }, { {1, 2, 3} }
- Bell number : S(n+1, k) = k*S(n, k) + S(n, k-1)
  - Bell(n, k) là số lượng cách chia tập {1, 2, …, n + 1} với tập con có tối đa k+1 phần tử
    VD. Bell(3, 2) = 3 vì tập {1,2,3,4} có 3 cách chia ra các tập con với số lượng phần tử tối đa là 3
    {{1}, {2, 4}, {3}}, {{1, 4}, {2}, {3}}, {{1, 2, 4}, {3}}
- Cách chia với tập N tương ứng số Bell(N,0)

```
1
1 2
2 3 5
5 7 10 15
15 20 27 37 52
```

Tam giác Bell từ 0

https://www.geeksforgeeks.org/bell-numbers-number-of-ways-to-partition-a-set/

Thuật toán quy hoạch động

- Bản bottom-up
- Thời gian và bộ nhớ đều cỡ $O(n^2)$

```
int bellNumber(int n)
{
    int bell[n+1][n+1];
    bell[0][0] = 1;
    for (int i=1; i<=n; i++)
    {
        // Explicitly fill for j = 0
        bell[i][0] = bell[i-1][i-1];


        // Fill for remaining values of j
        for (int j=1; j<=i; j++)
            bell[i][j] = bell[i-1][j-1] + bell[i][j-1];
    }
    return bell[n][0];
}
```

Thuật toán quy hoạch động

- Ví dụ 3. Tìm số cách tạo ra tập con k phần tử từ tập n phần tử ban đầu

$$P(n,k) = \frac{n!}{(n-k)!} = n(n-1)(n-2)\dots(n-k+1)$$
$$= P(n-1,k) + k * P(n-1,k-1)$$

```
int permutationCoeff(int n, int k)
{
    int P[n + 1][k + 1];
    for (int i = 0; i <= n; i++)
    {
        for (int j = 0; j <= std::min(i, k); j++)
        {
            // Base Cases
            if (j == 0)
                P[i][j] = 1;

             else
                P[i][j] = P[i - 1][j] +
                          (j * P[i - 1][j - 1]);


            P[i][j + 1] = 0;
        }
    }
    return P[n][k];
}
```

$T(n) = O(nk)$

```
int permutationCoeff(int n, int k)
{
    int fact[n + 1];
    fact[0] = 1;

    for(int i = 1; i <= n; i++)
    fact[i] = i * fact[i - 1];

    // P(n,k) = n! / (n - k)!
    return fact[n] / fact[n - k];
}
```

```
int PermutationCoeff(int n, int k)
{
    int P = 1;
    // Compute n*(n-1)*(n-2)....(n-k+1)
    for (int i = 0; i < k; i++)

        P *= (n-i) ;
    return P;
}
```

$T(n) = O(n)$

Thuật toán quy hoạch động

- Ví dụ 4. Bài toán lát sàn - tìm số cách lát sàn
  - Sàn nhà kích thước cỡ 2xn
  - Gạch kích thước cỡ 2x1
- Có bao nhiêu cách lát sàn khác nhau?
- Sàn 2x3 có 3 cách lát
  - 3 viên gạch xếp dọc
  - 1 dọc + 2 viên ngang
  - 2 ngang + 1 dọc
- Sàn 2x4 có 5 cách lát
  - Cả 1 gạch đều lát dọc
  - Cả 4 gạch đều xếp ngang
  - 2 dọc và 2 ngang
  - 2 ngang, 2 dọc
  - 1 dọc, 2 ngang, 1 dọc

https://www.geeksforgeeks.org/tiling-problem/

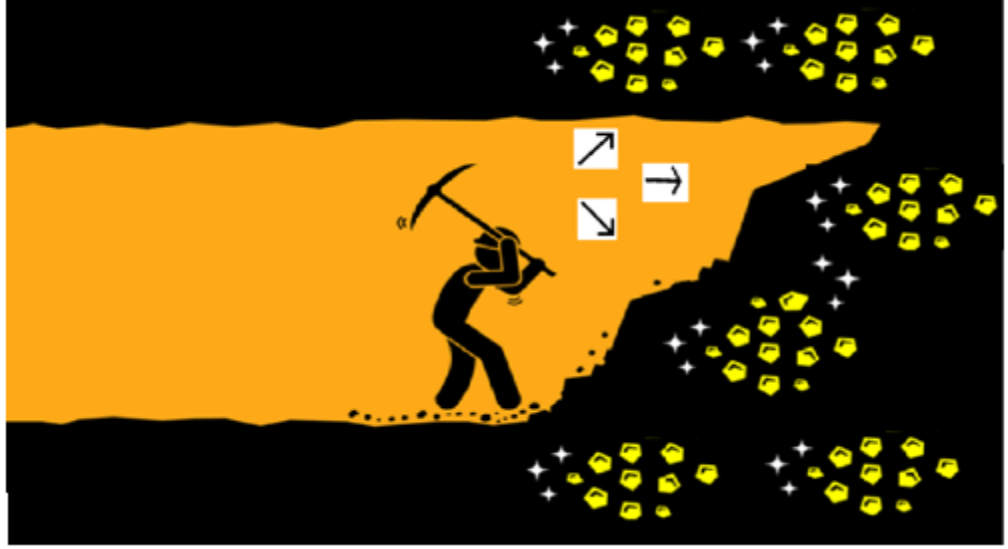
Bài toán lát sàn

- Trường hợp cơ sở
  - Sàn nhà kích thước 2x1 chỉ có 1 cách lát
  - Sàn 2x2 có 2 cách lát
- Tổng quát
  - Sàn nhà 2xn có số cách là count(n-1) + count(n-2)
  - Hoặc là 1 viên xếp dọc hoặc là 2 viên xếp ngang

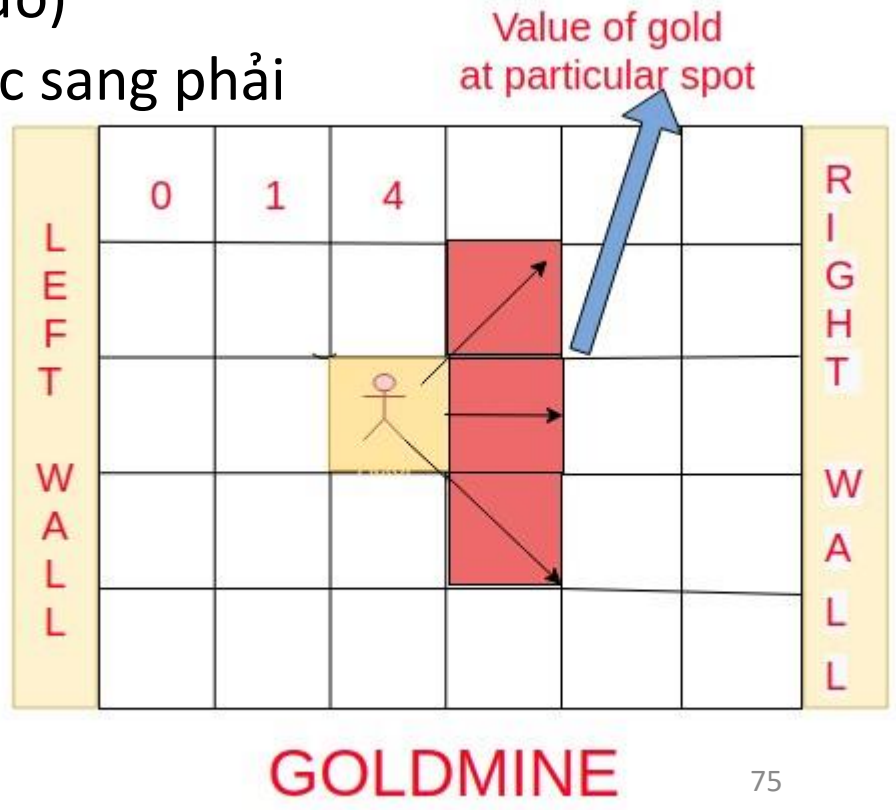
```
int count(int n)
{

   if (n == 0) return 0;
   if (n == 1) return 1;
   return count(n - 1) + count(n - 2);
}
```

```
int count(int n)
{
   if (n == 0) return 0;
   if (n == 1) return 1;
   int prev2=0,prev1=1,curr = 0;
   for (int i = 2; i <= n; i++)
   {
      curr = prev1 + prev2;
      prev2 = prev1;
      prev1 = curr;
   }
   return curr;
}
```

Thuật toán quy hoạch động



- Ví dụ 5: Bài toán thu tập phần thưởng, đào vàng
  - Đầu vào là ma trận mxn
  - Mỗi ô sẽ có 1 phần thưởng (hoặc lượng vàng nào đó)
  - Chỉ có thể đi theo hướng chéo lên, chéo xuống hoặc sang phải
  - Tìm tổng phần thưởng lớn nhất




```
Input: mat[][] = {
{1, 3, 1, 5},
{2, 2, 4, 1},
{5, 0, 2, 3},
{0, 6, 1, 2}};

Output : 16
(2,0) -> (1,1) -> (1,2) -> (0,3) hoặc
(2,0) -> (3,1) -> (2,2) -> (2,3)
```

Thuật toán quy hoạch động

```
Input: mat[][] = {
{1, 3, 1, 5},
{2, 2, 4, 1},
{5, 0, 2, 3},
{0, 6, 1, 2}};
```

- Tạo bảng tổng phần thưởng thu được của từng ô
  - Các ô lề bên trái: fill bằng giá trị thưởng hiện tại của ô đó
  - Các ô tiếp theo: tổng thưởng = giá trị của ô đó + MAX (phần thưởng trong các ô kề trước)

1			
2			
5			
0			

1	5		
2	7		
5	5		
0	11		

1	5	8	
2	7	11	
5	5	13	
0	11	12	

1	5	8	16
2	7	11	14
5	5	14	16
0	11	12	15

-			
-			
-			
-			

-	/		
-	/		
-	-		
-	\		

-	/	/	
-	/	-	
-	-	/	
-	\	-	

-	/	/	-
-	/	-	/
-	-	/	-
-	\	-	\

Thuật toán

```
int getMaxGold(int gold[][MAX], int m, int n)
{
    int goldTable[m][n];
    memset(goldTable, 0, sizeof(goldTable));


    for (int col=n-1; col>=0; col--)
    {
        for (int row=0; row<m; row++)
        {
            // Gold collected on going to the cell on the right(->)
            int right = (col==n-1)? 0: goldTable[row][col+1];
            // Gold collected on going to the cell to right up (/)
            int right_up = (row==0 || col==n-1)? 0: goldTable[row-1][col+1];


            // Gold collected on going to the cell to right down (\)
            int right_down = (row==m-1 || col==n-1)? 0: goldTable[row+1][col+1];


            // Max gold collected from taking either of the above 3 paths
            goldTable[row][col] = gold[row][col] + max(right, max(right_up, right_down));
        }
    }
    int res = goldTable[0][0];
    for (int i=1; i<m; i++) res = max(res, goldTable[i][0]);
    return res;
}
```

https://www.geeksforgeeks.org/gold-mine-problem/

Thuật toán quy hoạch động

- Ví dụ 6. Bài toán đổi tiền
  - Có m loại tờ tiền với mệnh giá khác nhau (số lượng không giới hạn) cho bởi mảng S
  - Cần đổi cho số tiền n, hỏi có bao nhiêu cách để đổi
- Với N = 4 và S = {1,2,3}, có 4 cách là : {1,1,1,1},{1,1,2},{2,2},{1,3}
- Với tờ tiền thứ k+1, mệnh giá S[k] trong danh sách?
  - Chọn tờ tiền thứ k mệnh giá S[k] thì số cách sẽ là số cách đổi của N-S[k]
  - Không chọn thì vẫn còn N tiền và chỉ còn lại m-1 loại tờ tiền
- Trường hợp cơ sở: N = 0 → Chỉ có 1 cách
- Công thức tìm: count(S, m - 1, n) + count(S, m, n - S[m - 1])

https://www.geeksforgeeks.org/coin-change-dp-7/

Thuật toán quy hoạch động

- Ví dụ 7. Đoạn con có tổng lớn nhất
  - Đầu vào 1 dãy số n phần tử
  - Tìm đoạn con (các phần tử phải liên tiếp) sao cho tổng giá trị các phần tử là lớn nhất
- Dãy ban đầu vào: {1, 4, 2, -5, 3, 1, -8, 6, -7, 4} → đoạn con lớn nhất là {1,4, 2}
- Gợi ý: khi tính tổng các phần tử lần lượt từ đầu dãy tới phần tử a[i]
  - Nếu tổng trước a[i] là S[i-1] <0 thì dãy S[i] sẽ bắt đầu lại (dãy con mới) tại a[i]
  - Ngược lại, dãy S[i] sẽ là dãy S[i-1] sẽ bao gồm luôn a[i]
  - Tổng lớn nhất? Là giá trị lớn nhất trong danh sách các tổng con S[i]
- A[] = {1, 4, 2, -5, 3, 1, -8, 6, -7, 4} thì S[] sẽ được điền như sau

1	5	7	2	5	6	-2	6	-1	4
S	+	+	+	+	+	+	S	+	S

Mảng đánh dấu, S là bắt đầu đoạn con, + là bao gồm phần tử a[i]
Trong mảng S thì giá trị lớn nhất là 7 tại i = 2, truy ngược lại mảng đánh dấu → dãy sẽ là {1,4,2}

$T(n) = O(n)$

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

Thuật toán quy hoạch động

- Ví dụ 8. Tìm đoạn con tăng dài nhất
  - Đầu vào 1 dãy số n phần tử
  - Tìm đoạn con (các phần tử phải liên tiếp) sao a[i] ≤a[i+1] và số lượng phần tử lớn nhất
- Gợi ý:
- Chia để trị: Chia đôi dãy:
    - Dãy tăng dài nhất hoặc nằm trọn ở nửa trái/phải
    - Hoặc nằm vắt ngang giữa dãy (1 phần bên nửa trái và 1 phần bên nửa phải, chứa phần tử giữa)
- Quy hoạch động
  - Đoạn con có 1 phần tử là dãy tăng nhỏ nhất
  - Với phần tử a[i] nếu a[i-1] ≤ a[i] thì đoạn tăng lớn nhất = đoạn tăng lớn nhất tại a[i-1] thêm 1
  - Ngược lại, bắt đầu dãy tăng độ dài 1 từ a[i]

Thuật toán quy hoạch động

- Ví dụ 9. Dãy tăng dài nhất
  - Đầu vào 1 dãy số n phần tử
  - Hãy loại đi một số phần tử để thu được dãy tăng có số lượng phần tử lớn nhất
- Gợi ý: Dùng cách tiếp cận đệ quy

arr[ ]	10	22	9	33	21	50	41	60	80
LIS	1	2		3		4		5	6

A[0]	A[1]	A[2]	..	A[i]	

LIS(i) là độ dài dãy tăng lớn nhất từ đầu tới vị trí a[i]
Dãy có 1 phần tử dãy tăng dài nhất là 1, vậy LIS(0)=1
LIS(1) = ?

- Nếu A[1]>= A[0] thì LIS(1) = LIS(0) +1
- Ngược lại LIS(1) = 1

LIS(2) =?

- Nếu A[2]>= A[1] thì LIS(2) = LIS(1) +1
- Nếu A[2]>= A[0] thì LIS(2) = LIS(0) + 1
- Ngược lại? LIS(2) = 1

Vậy LIS(i) = MAX(1+L[j]) nếu A[j] <= A[i] với 0<=j<i
hoặc 1 nếu ngược lại (dãy giảm dần)

$$T(n) = ?$$

Dãy tăng dài nhất

A	1	5	2	4	7	5	6	8

- Các bài toán con bị lặp nhiều?
  - Dùng quy hoạch động theo mô hình bottom-up
  - Điền mảng LIS từ trái qua
  - Khởi tạo các giá trị L[] = {1}

L	1	1	1	1	1	1	1	1

Khởi tạo

L	1	2	1	1	1	1	1	1

A[1]>A[0] nên L[1] = L[0]+1 = 2

L	1	2	2	1	1	1	1	1

A[2]>A[0] nên L[2] = L[0]+1 = 2
A[2]<A[1] nên L[2] không đổi

L	1	2	2	3	1	1	1	1

A[3]>A[0] nên L[3] = L[0]+1 = 2
A[3]<A[1] nên Không đổi
A[3]>A[2] nên L[3] = L[2]+1=3

L	1	2	2	3	4	1	1	1

A[4]>A[0] nên L[4] = L[0]+1 = 2
A[4]>A[1] nên L[4] = L[1]+1 = 3
A[4]>A[2] nên L[4] = L[2]+1 = 3
A[4]>A[3] nên L[4] = L[3]+1 = 4

L	1	2	2	3	4	4	5	6

Thuật toán quy hoạch động

- Bài tập 1. Xây dựng hàm để khớp xâu tìm kiếm
  - ? Thay thế cho 1 ký tự
  - * thay thế cho 1 tập rỗng hoặc n ký tự
  - isMatching("abab","abab") → true
  - isMatching("abab","a**b") → true
  - isMatching("ababab","ab*b") → true
  - isMatching("","*") → true
  - isMatching("aaaaaab","*?*b") → true

Thuật toán quy hoạch động

- Bài tập 2. Tìm dãy palindromes dài nhất tạo thành từ xâu đầu vào
  VD. "abfdRacecaRAbAorTITabcdef"

- Bài tập 3. Tìm dãy con chung có độ dài lớn nhất - LCS của 2 xâu (các ký tự trong xâu con cần đúng thứ tự so với xâu gốc)
  Xâu “ABCDGH” và “AEDFHR” có xâu con chung dài nhất “ADH” độ dài 3.

  Xâu “AGGTAB” và “GXTXAYB” có xâu con chung dài nhất “GTAB” độ dài 4.

https://www.geeksforgeeks.org/dynamic-programming