Qui hoạch động

Nguyên lý của sự tối ưu

"Lời giải tối ưu cho thực thể bất kỳ của bài toán tối ưu được hình thành từ việc tổ hợp các lời giải tối ưu cho những thực thể nhỏ hơn".

Kỹ thuật này hướng đến việc giải quyết những vấn đề mang bản chất đệ qui chia để trị, khi tồn tại quan hệ truy hồi giữa lời giải của bài toán ban đầu và lời giải của các bài toán con cùng kiểu.

Ví dụ: Tìm số thứ n của dãy Fibonacci.

Ví dụ: Tính hệ số nhị thức $\binom{n}{k}$.

Hệ số nhị thức được xác định bởi công thức sau:

$$\binom{n}{k} = \frac{n!}{k!\,(n-k)!}\ \text{với}\ 0 \le k \le n$$

và tính đệ qui được thể hiện bởi công thức:

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k} & 0 < k < n \\ 1 & k = 0 \lor k = n \end{cases}$$

Công thức tổng quát là:

$$\binom{i}{j} \Leftrightarrow C[i,j] = \begin{cases} C[i-1,j-1] + C[i-1,j] & 0 < j < i \\ 1 & j = 0 \lor j = i \end{cases}$$

	0	1	2	…	$j-1$	j	…	$k-1$	k
0	1								
1	1	1							
2	1	2	1						
⋮									
$i-1$					$C[i–1, j–1]$	$C[i–1, j]$			
i						$C[i, j]$			
⋮									
k	1								1
⋮									
$n–1$	1							$C[n–1, k–1]$	$C[n–1, k]$
n	1								$C[n, k]$

Giải thuật (đệ qui)

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

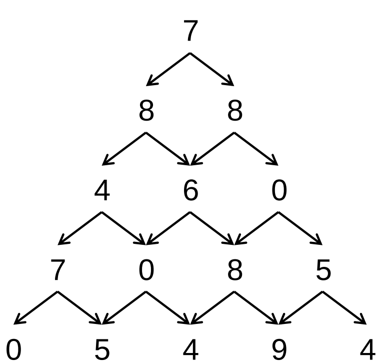
Giải thuật

```
Binomial(n, k) {
   C[0 .. n, 0] = C[0 .. k, 0 .. k] = 1;
   for (i = 1; i ≤ n; i++)
      for (j = 1; j < (i ≤ k ? i : k + 1); j++)
         C[i, j] = C[i – 1, j – 1] + C[i – 1, j];
   return   C[n, k];
}
```

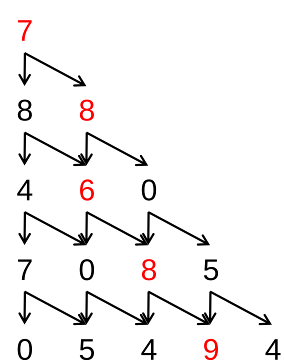
Đánh giá: $A(n, k) \in \Theta(nk)$

Ví dụ: Bài toán tam giác

Phát biểu: Cho trước tam giác "đều" cạnh n chứa các số nguyên dương. Tính tổng lớn nhất của một con đường đi từ đỉnh đến đáy và chỉ ra con đường này. Yêu cầu là mỗi bước đi xuống lệch sang trái hoặc phải một vị trí).



(a)



(b)

	0	1	2	3	4	5
0	0	0				
1	0	7	0			
2	0	15	15	0		
3	0	19	21	15	0	
4	0	26	21	29	20	0
5	0	26	31	33	38	24

(c)

Gọi $S(i, j)$ là tổng lớn nhất của đoạn đường từ đỉnh đến vị trí dòng i và cột j.

$$S(i,j) = \begin{cases} \max\big(S(i-1,j-1), S(i-1,j)\big) + a_{i,j} & i \neq j \wedge i,j \neq 1 \\ a_{i,j} & i = j = 1 \\ S(i-1,j-1) + a_{i,j} & i = j \wedge i,j \neq 1 \\ S(i-1,j) + a_{i,j} & i \neq j \wedge j = 1 \end{cases}$$

Sử dụng mảng S hai chiều $(n + 1) \times (n + 1)$ để tính toán mọi con đường có thể xuất phát từ đỉnh. Ban đầu, $S[0..n, 0] = 0, S[j, j + 1] = 0$ với $0 \leq j \leq n - 1$.

Giải thuật

```
… Đưa mảng a vào bảng S …

// Xây dựng bảng S
S[0 .. n, 0] = S[0 .. n − 1, 1 .. n] = 0;
for (i = 1; i ≤ n; i++)
   for (j = 1; j ≤ i; j++)
      S[i][j] = max(S[i - 1][j - 1], S[i-1][j]) + a[i][j];
return   "Phần tử lớn nhất trên dòng n"
```

Đánh giá: $S(n) \in \Theta(n^2)$

Mở rộng: Xác định con đường dẫn đến kết quả

Bài toán đổi tiền xu

Gọi mệnh giá k đồng xu là $\{x_1, x_2, \ldots, x_k\}$ và giả định rằng, đồng xu có mệnh giá nhỏ nhất là 1 xu để đảm bảo luôn có lời giải.

Gọi $C(n)$ là số lượng đồng xu tối thiểu để đổi n xu:

$$C(n) = 1 + \min_{1 \le i \le k} \{C(n - x_i)\} \text{ với } n \ge x_i, C(0) = 0$$

Giải thuật

```
int   changeCoinsDP(int coins[], int k, int money) {
   int C[0 .. money] = 0;

   for (cents = 1; cents ≤ money; cents++) {
      int minCoins = cents;
      for (i = 1; i ≤ k; i++) {
         if (coins[i] > cents)
            continue;
         if (C[cents - coins[i]] + 1 < minCoins)
            minCoins = C[cents - coins[i]] + 1;
      }
      C[cents] = minCoins;
   }
   return C[money];
}
```

Đánh giá: $\Theta(kn)$

Mở rộng: Xác định các đồng xu trong lời giải.

Tìm dãy con tăng nghiêm ngặt dài nhất

Phát biểu: Cho dãy số (nguyên dương) $a_1, a_2, \dots, a_n$. Một dãy con tăng nghiêm ngặt được hình thành bằng cách loại bỏ không hoặc nhiều phần tử nào đó để những phần tử còn lại $a_{i_1}, a_{i_2}, \dots, a_{i_k}$ với $1 \le k \le n$ thỏa điều kiện $a_{i_p} < a_{i_q}$ với $1 \le p < q \le k$. Tìm dãy con tăng nghiêm ngặt dài nhất.

Gọi $L(i)$ là chiều dài của dãy con tăng nghiêm ngặt dài nhất, *với a_i là phần tử cuối cùng trong dãy này và tất nhiên có giá trị lớn nhất trong dãy*.

$$L(i) = 1 + \max_{1 \le j < i:\, a_j < a_i} \{L(j)\}$$

với $L(1) = 1$

Giải thuật (đệ qui)

```
lis(a[1 .. n], i) {
   if (i == 1)
      return 1;

   tmpMax = 1;
   for (j = 1; j < i; j++)
      if (a[j] < a[i]) {
         res = lis(a, j);
         if (res + 1 > tmpMax)
            tmpMax = res + 1;
      }
   if (max < tmpMax) // global variable
      max = tmpMax;
   return   tmpMax;
}

for (i = 1; i ≤ n; i++)
   lis(a, i);
print(max);
```

Đánh giá: Độ phức tạp của giải thuật phụ thuộc vào phân bố của dữ liệu đầu vào. Để đơn giản, cho rằng các phần tử dữ liệu khác nhau từng đôi một. Gọi $T(i)$ là chi phí thực thi của hàm `lis(a, i)`, $\forall i \in [1, n]$.

- Trường hợp xấu nhất: $\Theta(2^n)$
- Trường hợp tốt nhất: $\Theta(n^2)$
- Trường hợp trung bình: Không dễ để xác định do phụ thuộc vào kết quả của phép so sánh cơ sở.

Với tiếp cận qui hoạch động, gọi $L[i]$ là chiều dài dài nhất của dãy con tăng nghiêm ngặt có a_i là phần tử cuối cùng (thuộc về dãy con này). Nhận thấy:

$$L[i] = \max_{1 \le j < i:\, a_j < a_i} \{L[j]\} + 1$$

Giải thuật

```
lis_DP(a[1..n]) {
   L[1 .. n] = 1;
   for (i = 2; i ≤ n; i++)
      for (j = 1; j < i; j++)
         if ((a[j] < a[i]) && (L[j] + 1 > L[i]))
            L[i] = L[j] + 1;
   return   "Phần tử lớn nhất trên mảng L";
}
cout << lis_DP(a);
```

Đánh giá: $\Theta(n^2)$

Mở rộng: Chỉ ra dãy con tăng tuyệt đối dài nhất.

Tìm dãy con chung dài nhất

Phát biểu: Cho hai chuỗi ký tự $S = s_1 s_2 \dots s_m$ và $T = t_1 t_2 \dots t_n$. Tìm dãy con chung dài nhất (và chiều dài của nó) xuất hiện trong cả hai chuỗi trên. Nói cách khác, đây là dãy các vị trí trong S: $1 \le i_1 < i_2 < \cdots < i_k \le m$ và trong T: $1 \le j_1 < j_2 < \cdots < j_k \le n$, sao cho ký tự $s_{i_h} \equiv t_{j_h}, \forall h \in [1, k]$ và k là lớn nhất.

Ví dụ: $S = XYXZPQ, T = YXQYXP$. Dãy con chung dài nhất là $XYXP$ với độ dài 4, vì $S = XYXZPQ, T = YXQYXP$.

Gọi $L(i, j)$ là chiều dài của *dãy con chung dài nhất* nằm bên trong hai *chuỗi con* $s_1 s_2 \dots s_i$ và $t_1 t_2 \dots t_j$.

- Nếu $s_i = t_j$: $L(i, j) = 1 + L(i-1, j-1)$
- Nếu $s_i \ne t_j$: $L(i, j) = \max\{L(i-1, j), L(i, j-1)\}$

với $L(i, 0) = L(0, j) = 0$.

Giải thuật (đệ qui)

```
int   LCS(char S[], int i, char T[], int j) {
   if ((i == 0) || (j == 0))
      return 0;

   if (S[i] == T[j])
      return   1 + LCS(S, i - 1, T, j - 1);
   else
      return   max(LCS(S, i - 1, T, j), LCS(S, i, T, j - 1));
}
cout << LCS(S, m, T, n);
```

Đánh giá: $\Theta(2^n)$

Với qui hoạch động, sử dụng bảng hai chiều L kích thước $m \times n$ để lưu trữ kết quả của những thực thể của bài toán con.

Ô $L[i,j]$ chứa chiều dài của *dãy con chung dài nhất* nằm trên hai *chuỗi con* $s_1 s_2 \dots s_i$ và $t_1 t_2 \dots t_j$.

- Nếu $s_i = t_j$: $L[i,j] = L[i-1,j-1] + 1$
- Nếu $s_i \neq t_j$: $L[i,j] = \max\{L[i-1,j], L[i,j-1]\}$

với $L[0,j] = L[i,0] = 0$. Rõ ràng, $L[m,n]$ chứa chiều dài cần tìm.

Giải thuật

```
LCS_Dyn(char S[], int m, char T[], int n) {
   L[1 .. m][0] = L[0][1 .. n] = 0

   for (i = 1; i ≤ m; i++)
      for (j = 1; j ≤ n; j++)
         if (S[i] == T[j])
            L[i][j] = 1 + L[i - 1][j - 1];
         else
            L[i][j] = max(L[i - 1][j], L[i][j - 1]);
   return   L[m][n];
}
```

Đánh giá: Chi phí của giải thuật là $\Theta(mn)$.

Mở rộng: Tìm dãy con chung.

Giải thuật (của) Floyd tìm đường đi ngắn nhất giữa tất cả các cặp đỉnh

Phát biểu: Cho đồ thị *liên thông* có trọng số (có hướng hay vô hướng). Tìm khoảng cách ngắn nhất từ mỗi đỉnh đến mọi đỉnh còn lại.

Cho rằng đồ thị có n đỉnh. Sử dụng mảng hai chiều D (gọi là *ma trận khoảng cách*) kích thước $n \times n$ với phần tử d_{ij} là chiều dài ngắn nhất đi từ đỉnh nhãn i đến đỉnh nhãn j $(1 \leq i \neq j \leq n)$.

Giải thuật (của) Floyd sẽ tính toán ma trận D thông qua dãy ma trận:

$$D^{(0)}, D^{(1)}, \dots, D^{(k-1)}, D^{(k)}, \dots, D^{(n)} \equiv D$$

Phần tử $d_{ij}^{(k)} \in D^{(k)}$ (với $k = 0,1, \dots, n$) là chiều dài con đường ngắn nhất (trong tất cả các con đường) đi từ đỉnh nhãn i đến đỉnh nhãn j sao cho, các đỉnh trung gian (nếu có) được đánh số từ k trở xuống.

$$d_{ij}^{(k)} = \min_{1 \leq k \leq n} \left\{ d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \right\}$$

Giải thuật

```
Floyd(W[1 .. n, 1 .. n]) { // Ma trận kề chứa trọng số
   D(0) = W;
   for (k = 1; k ≤ n; k++)
      for (i = 1; i ≤ n; i++)
         for (j = 1; j ≤ n; j++)
            D(k)[i, j] = min{D(k-1)[i, j], D(k-1)[i, k] + D(k-1)[k, j]};
   return   D(n);
}
```

Đánh giá: $\Theta(n^3)$

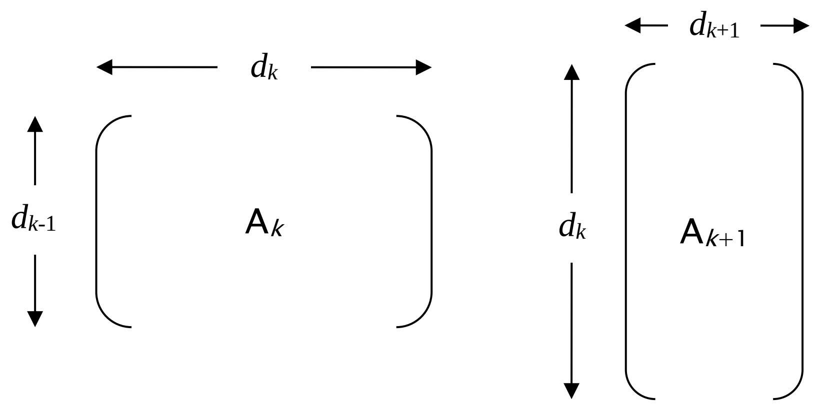
Nhân dãy ma trận

Phát biểu: Xác định thứ tự để nhân dãy n ma trận $A_1 \times A_2 \times \dots \times A_n$ có kích thước $d_0 \times d_1, d_1 \times d_2, \dots, d_{n-1} \times d_n$ với chi phí thấp nhất.

Ví dụ: Nhân 4 ma trận $A \times B \times C \times D$ với kích thước lần lượt là $50 \times 20, 20 \times 1, 1 \times 10, 10 \times 100$. So sánh *ba* trong số *năm* thứ tự nhân 4 ma trận nêu trên:

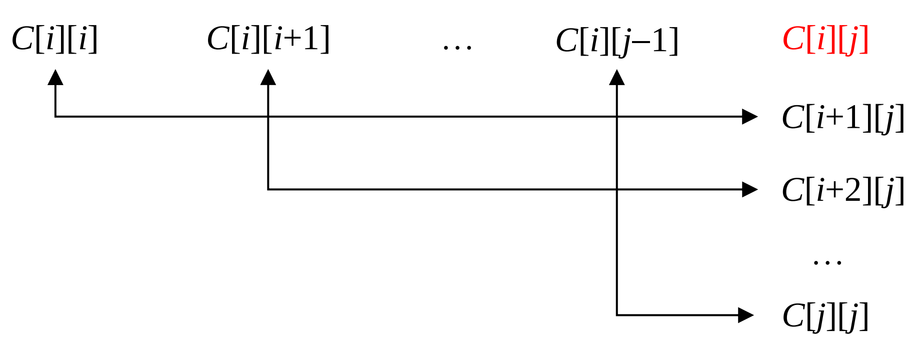
Thứ tự nhân	Số phép tính
$A \times \big((B \times C) \times D\big)$	$20 \times 1 \times 10 + 20 \times 10 \times 100 + 50 \times 20 \times 100 = 120200$
$\big(A \times (B \times C)\big) \times D$	$20 \times 1 \times 10 + 50 \times 20 \times 10 + 50 \times 10 \times 100 = 60200$
$(A \times B) \times (C \times D)$	$50 \times 20 \times 1 + 1 \times 10 \times 100 + 50 \times 1 \times 100 = 7000$

Qui ước: Nếu $A_k \times A_{k+1}$ với $1 \le k < n$ thì kích thước của ma trận A_k và A_{k+1} lần lượt là $d_{k-1} \times d_k$ và $d_k \times d_{k+1}$ và kích thước ma trận tích là $d_{k-1} \times d_{k+1}$.



Xét tích $A_i \times A_{i+1} \times \dots \times A_j$ với $1 \le i \le j \le n$. Gọi $C(i,j)$ là chi phí tối thiểu của dãy phép nhân này.

$$C(i,j) = \begin{cases} \min\limits_{i \le k < j} \{C(i,k) + C(k+1,j) + d_{i-1} \times d_k \times d_j\} & i < j \\ 0 & i = j \end{cases}$$



Giải thuật

```
ChainMatrixMult(d[0 .. n], P[1 .. n][1 .. n]) {
   C[1 .. n, 1 .. n] = 0;

   for (diag = 1; diag < n; diag++)
      for (i = 1; i ≤ n - diag; i++) {
         j = i + diag;
         C[i, j] = min_{i≤k<j}{C[i,k] + C[k+1,j] + d[i−1] × d[k] × d[j]};
         P[i, j] = Gia tri k tim duoc;
      }

   return C[1, n];
}
```

Đánh giá: $\Theta(n^3)$

Mở rộng:

Mảng P hai chiều có nhiệm vụ lưu lại giá trị phân tách k khiến cho tích của dãy ma trận từ A_i đến A_j là nhỏ nhất.

Giải thuật

```
order(i, j) {
   if (i == j)
      cout << "A" <<  i;
   else {
      k = P[i][j];
      cout << "(";
      order(i, k);
      order(k + 1, j);
      cout << ")";
   }
}

order(1, n);
```

Cây nhị phân tìm kiếm tối ưu

Cây nhị phân tìm kiếm cổ điển xem các khóa tìm kiếm là như nhau. Nếu thu thập được thông tin về tần suất mỗi khóa được tìm kiếm thì để tăng hiệu quả truy xuất:

- Đặt những khóa được tìm kiếm nhiều ở gần nút gốc,
- Đặt những khóa càng ít tìm càng nằm xa.

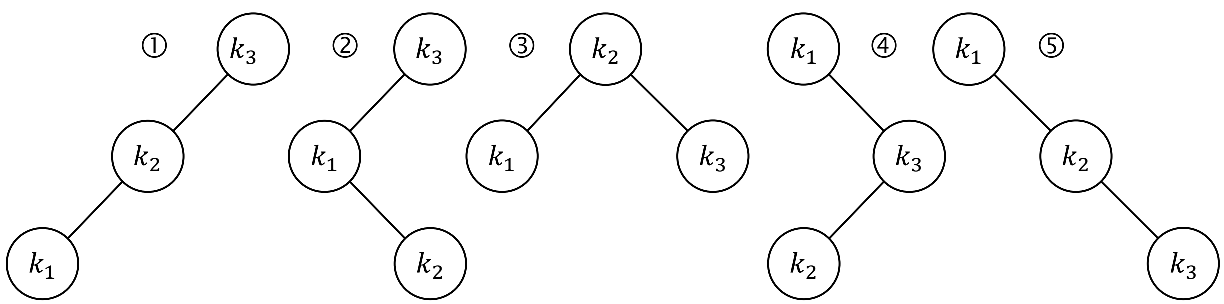
Như vậy, chi phí trung bình cho việc tìm kiếm sẽ là tối thiểu. Cây có tính chất này gọi là Cây nhị phân tìm kiếm tối ưu.

Qui ước: Gọi

- $k_1, k_2, \dots, k_n$ là khóa của n nút trên cây. Cho rằng $k_1 < k_2 < \cdots < k_n$
- p_i là xác suất để k_i trở thành khóa tìm kiếm
- c_i là số phép so sánh để tìm ra k_i:
$$c_i = level(k_i) + 1$$
- Chi phí chung để tìm một khóa bất kỳ trên cây là
$$Cost = \sum_{i=1}^{n} (c_i \times p_i)$$

và chúng ta cần xây dựng cây sao cho giá trị này là nhỏ nhất.

Ví dụ: Xét cây nhị phân tìm kiếm có ba nút, với các xác suất lần lượt là $p_1 = 0.7, p_2 = 0.2, p_3 = 0.1$. Khóa các nút là: $k_1 < k_2 < k_3$. Có 5 khả năng hình thành cây:



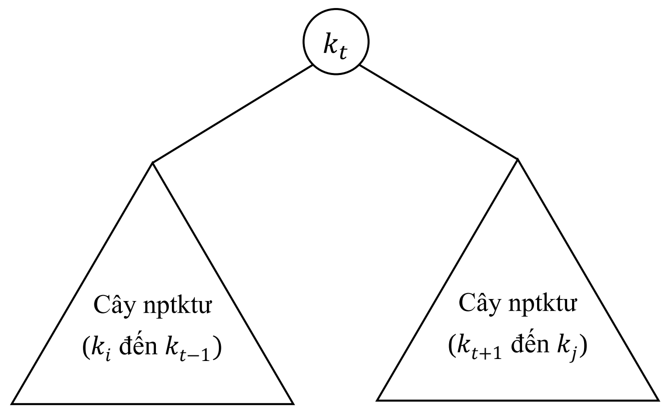
1. 3 (0.7) + 2 (0.2) + 1 (0.1) = 2.6
2. 2 (0.7) + 3 (0.2) + 1 (0.1) = 2.1
3. 2 (0.7) + 1 (0.2) + 2 (0.1) = 1.8
4. 1 (0.7) + 3 (0.2) + 2 (0.1) = 1.5
5. 1 (0.7) + 2 (0.2) + 3 (0.1) = **1.4**

Gọi $C(i,j)$ là chi phí chung nhỏ nhất để tìm một khóa bất kỳ trên cây nhị phân tìm kiếm chứa các khóa từ k_i đến k_j. Cây nhị phân này được gọi là $T_{i,j}$ với $1 \leq i \leq j \leq n$.

- Nếu $i = j$: Cây $T_{i,j}$ chỉ có một nút.

$$C(i,i) = c_i \times p_i = 1 \times p_i = p_i$$

- Nếu $i > j$: Cây $T_{i,j}$ được xem là rỗng và $C(i,j) = 0$.
- Nếu $i < j$: Quan tâm đến mọi khả năng có thể để hình thành nên một cây $T_{i,j}$. Gọi $T_{i,j}^t$ là cây $T_{i,j}$ có gốc chứa khóa k_t nào đó với $i \leq t \leq j$.



$$C(i,t-1) = \sum_{s=i}^{t-1} (\#\, So\ sánh\ tìm\ key_s) \times p_s = \sum_{s=i}^{t-1} c_s \times p_s$$

$$C(t+1,j) = \sum_{s=t+1}^{j} (\#\, So\ sánh\ tìm\ key_s) \times p_s = \sum_{s=t+1}^{j} c_s \times p_s$$

Cây $T_{i,j}^t$ có chi phí tìm kiếm chung là:

$$\left(C(i,t-1) + \sum_{s=i}^{t-1} p_s\right) + \left(C(t+1,j) + \sum_{s=t+1}^{j} p_s\right) + p_t$$

$$= C(i,t-1) + C(t+1,j) + \sum_{s=i}^{j} p_s$$

$$\Rightarrow C(i,j) = \min_{i \leq t \leq j} \left\{ C(i,t-1) + C(t+1,j) + \sum_{s=i}^{j} p_s \right\}$$

$$= \min_{i \leq t \leq j} \{C(i,t-1) + C(t+1,j)\} + \sum_{s=i}^{j} p_s$$

với $1 \leq i \leq j \leq n$.

Ví dụ: Xét bảng C kích thước $(1..n+1) \times (0..n)$ với $n = 10$.

$i \downarrow$ \ $j \rightarrow$	0	1	2	3	4	5	6	7	8	9	10
1	0	p_1									?
2		0	p_2								
3			0	p_3							
4				0	p_4			$C(4,7)$			
5					0	p_5					
6						0	p_6				
7							0	p_7			
8								0	p_8		
9									0	p_9	
10										0	p_{10}
11											0

Giải thuật

```
OptimalBST(p[1..n]) {
   int C[1 .. n + 1, 0 .. n], R[1 .. n + 1, 0 .. n], ;
   C[1 .. n + 1, 0 .. n] = R[1 .. n + 1, 0 .. n] = 0;
   C[1 .. n, 1 .. n] = p[1 .. n];
   R[1 .. n, 1 .. n] = 1 .. n;
   for (diag = 1; diag < n; diag++)
      for (i = 1; i ≤ n − diag; i++) {
         j = i + diag;
         minval = min_{i≤t≤j}(C[i, t − 1] + C[t + 1, j]);
         R[i, j] = Giá trị t tìm được;
         C[i, j] = minval + ∑_{s=i}^{j} p[s];
      }
   return      <C[1, n], R>;
}
```

Đánh giá: $\Theta(n^3)$

Giải thuật (Xây dựng cây)

```
tree(i, j) {
   t = R[i, j];
   if (t == 0)    return NULL;
   p         = new node;
   p->key    = key[t];
   p->left   = tree(i, t - 1);
   p->right  = tree(t + 1, j);
   return p;
}
root = tree(1, n);
```

Tổng các tập con

Phát biểu: Tìm tập con của tập đã cho $A = \{a_1, a_2, \dots, a_n\}$ gồm n số nguyên dương sao cho tổng các phần tử của tập con này bằng với k.

Giả sử đã tồn tại hàm `SubsetSums(A, k)` kiểu `boolean` để tìm kiếm sự tồn tại của tập $S \subseteq A$ sao cho tổng các phần tử của tập con này bằng k. Chọn một phần tử bất kỳ trong A, gọi là a:

i. Nếu $a > k$: Tiếp tục tìm kiếm với `SubsetSums(A\{a}, k)`.

ii. Ngược lại: câu trả lời sẽ là một trong hai khả năng sau:

   a. Cho rằng phần tử $a \in S$: Tiếp tục tìm kiếm với `SubsetSums(A\{a}, k-a)`.
   b. Cho rằng phần tử $a \notin S$: Tiếp tục tìm kiếm với `SubsetSums(A\{a}, k)`.

Điều kiện để quá trình đệ qui kết thúc là như sau:

- Nếu $k = 0$ (còn tập A có thể là rỗng hoặc không): Tồn tại tập S.
- Nếu $A = \emptyset$ và $k \neq 0$: Không tồn tại S.

Giải thuật (đệ qui)

```
SubsetSums(a[], n, k) {
   if (k == 0)
      return      true;
   if (n == 0)
      return      false;
   if (a[n] > k)
      return      SubsetSums(a, n - 1, k);
   return   SubsetSums(a, n - 1, k - a[n]) || SubsetSums(a, n - 1, k);
}
SubsetSums(a, n, k);
```

Đánh giá: $O(2^n)$

Với qui hoạch động, bảng $V[0..n, 0..k]$ sẽ được dùng để lưu giá trị:

- Nếu tồn tại *tập con* nào đó *của tập* $\{a_1, a_2, \dots, a_i\}$ có tổng là j $(1 \leq j \leq k)$:
$$V[i,j] = 1$$
- Ngược lại: $V[i,j] = 0$.

$$V[i,j] = \begin{cases} 1 & V[i-1,j] = 1 \lor V[i-1, j-a_i] = 1 \text{ với } j \geq a_i \\ 0 & \neq \end{cases}$$

với $V[0,1..k] = 0, V[0..n,0] = 1$.

Giải thuật

```
SubsetSumsDP(a[1 .. n], n, k) {
   int V[0 .. n, 0 .. k];

   V[0 .. n, 0] = 1;
   V[0, 1 .. k] = 0;
   for (i = 1; i ≤ n; i++)
      for (j = 1; j ≤ k; j++) {
         tmp = 0;
         if (j ≥ a[i])
            tmp = V[i – 1, j - a[i]];
         V[i, j] = V[i – 1, j] || tmp;
      }
}
SubsetSumsDP(a, n, k);
```

Đánh giá: Chi phí của giải thuật là $\Theta(nk)$.

Giải thuật (In kết quả)

```
if (V[n, k]) {
   while (k) {
      if (V[n – 1, k - a[n]] == 1 && V[n – 1, k] == 0) {
         cout << a[n] << " ";
         k -= a[n];
      }
      n--;
   }
}
```

Mở rộng 1

Phát biểu: Cho tập $A = \{a_1, a_2, \dots, a_n\}$ (có thể bằng nhau) gồm n số *nguyên dương*. Cho biết, tập này có thể chia thành hai tập con A_1 và A_2 sao cho: $A_1 \cap A_2 = \emptyset, A_1 \cup A_2 = A$ và tổng của các phần tử của hai tập *bằng nhau*.

Mở rộng 2

Phát biểu: Cho tập $A = \{a_1, a_2, \dots, a_n\}$ (có thể bằng nhau) gồm n số *nguyên dương*. Tìm cách chia tập này thành hai tập con A_1 và A_2 thỏa những điều kiện sau:

1. $A_1 \cap A_2 = \emptyset, A_1 \cup A_2 = A$
2. Nếu gọi S_A là tổng của các phần tử của tập A thì $\left|S_{A_1} - S_{A_2}\right|$ là nhỏ nhất.

Bài toán túi xách 0/1

Phát biểu: Cho n đồ vật có cân nặng là $w_1, w_2, \dots, w_n (\in \mathbb{Z}^+)$ và giá trị là $v_1, v_2, \dots, v_n (\in \mathbb{R}^+)$. Khả năng chứa của túi là $W (\in \mathbb{Z}^+)$. Tìm tập con có giá trị nhất của các đồ vật mà túi có thể mang được.

Gọi $T(i, j)$ là giá trị của tập con đáng giá nhất hình thành từ việc lấy một/nhiều/tất cả đồ vật trong số i đồ vật đầu tiên và cho *được* vào túi có sức chứa j.

$$T(i,j) = \begin{cases} \max\{T(i-1,j), v_i + T(i-1, j-w_i)\} & j \geq w_i \\ T(i-1,j) & j < w_i \end{cases}$$

$$\begin{cases} T(0,j) = 0 & j \geq 0 \\ T(i,0) = 0 & i \geq 0 \end{cases}$$

Ví dụ: Xét tập gồm 4 đồ vật: $\{w_1 = 2, v_1 = 12đ\}, \{w_2 = 1, v_2 = 10đ\}, \{w_3 = 3, v_3 = 20đ\}, \{w_4 = 2, v_4 = 15đ\}$ và $W = 5$.

i↓j→	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0					
3	0					
4	0					

i↓j→	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0					
4	0					

i↓j→	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0					

i↓j→	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	10	12	22	22	22
3	0	10	12	22	30	32
4	0	10	15	25	30	37

Giải thuật

```
Knapsack(w[1 .. n], v[1 .. n], W) {
   int T[0 .. n, 0 .. W];
   T[0 .. n, 0] = T[0, 1 .. W] = 0;
   for (i = 1; i ≤ n; i++)
      for (j = 1; j ≤ W; j++)
         if (j ≥ w[i])
            T[i, j] = max{T[i - 1, j], v[i] + T[i - 1, j - w[i]]};
         else
            T[i, j] = T[i - 1, j];
   return   T[n, W];
}
```

Mở rộng: Xác định các thành phần của tập con.

Bài toán đường đi người bán hàng

Cho rằng, đồ thị liên thông $G = (V, E)$ có tập các đỉnh là $V = \{v_1, v_2, \dots, v_n\}$. Không mất đi tính tổng quát, gọi v_1 là đỉnh bắt đầu của mọi chu trình.

Nhận xét:

"Nếu v_i là đỉnh đầu tiên theo ngay sau v_1 trong một chu trình *tối ưu* thì đoạn đường còn lại của chu trình này, đi từ v_i quay về v_1, sẽ phải là con đường *ngắn nhất* đi qua mọi đỉnh còn lại đúng một lần".

Gọi:

- W: Ma trận kề biểu diễn G. Phần tử $W[i, j]$ có giá trị k (trọng số của cạnh nối từ v_i đến v_j), giá trị ∞ (không có cạnh nối) hoặc 0 (khi $i = j$).
- $A(\subseteq V)$: Tập các đỉnh.
- $D[v_i, A]$: Chiều dài của con đường ngắn nhất từ v_i về đến v_1, đi qua mọi đỉnh trong tập A đúng một lần.

Một cách tổng quát, $\forall i \in [2, n], v_i \notin A$:

$$D[v_i, A] = \begin{cases} \min\limits_{j: v_j \in A} \{W[i, j] + D[v_j, A \setminus \{v_j\}]\} & A \neq \emptyset \\ W[i, 1] & A = \emptyset \end{cases}$$

Giải thuật

```
TSP(n, W[1 .. n, 1 .. n], P[1 .. n, 1 .. n]) {
   D[1 .. n, Tập con của V \ {v₁}];
   P[1 .. n, Tập con của V \ {v₁}];

   D[2 .. n, ∅] = W[2 .. n, 1];
   for (k = 1; k ≤ n - 2; k++)       // Kích thước của tập A
      for (mỗi tập A (⊆ V \ {v₁}) chứa k đỉnh)
         for (i: i ≠ 1 và vᵢ ∉ A) {
            D[i, A] = min_{j:v_j∈A} {W[i, j] + D[j, A \ {vⱼ}]};
            P[i, A] = giá trị j tìm được;
         }
   D[1, V \ {v₁}] = min_{2≤j≤n} {W[1][j] + D[j][V \ {v₁, vⱼ}]};
   P[1, V \ {v₁}] = giá trị j tìm được;
   return   { D[1, V \ {v₁}], P };
}
```

Đánh giá: $\Theta(n2^n)$

Mở rộng: Chỉ ra chu trình tối ưu

Giải thuật

```
TSP_Tour(P[1..n, 1..n]) {
   cout << v1;
   A = V \ {v1};
   k = 1;
   while (A ≠ ∅) {
      k = P[k, A];
      cout << vk;
      A = A \ {vk};
   }
}
```

Memoization

Một kỹ thuật lai giữa Chia để trị và Qui hoạch động để giải bài toán tối ưu. Cơ sở của kỹ thuật là sự kết hợp ưu điểm của hai tiếp cận truyền thống:

- Lối tư duy trực quan từ trên xuống (*top-down, depth-first analysis*) của Chia để trị thông qua cài đặt đệ qui.
- Tiếp cận từ dưới lên (*bottom-up, breadth-first analysis*) bằng cách lưu trữ kết quả các bài toán con của Qui hoạch động, sử dụng kỹ thuật lặp.

Ví dụ: Tìm số thứ n của dãy Fibonacci

Giải thuật

```
Fib(f[0 .. n], n) {
   if (f[n] < 0)
      f[n] = Fib(f, n - 1) + Fib(f, n - 2);
   return   f[n];
}
Fib_Memo(n) {
   f[0] = 0;
   f[1] = 1;
   f[2 .. n] = -1;
   return   Fib(f, n);
}
```

Ví dụ: Nhân dãy ma trận.

Hệ thức truy hồi của bài toán được chỉ ra như sau:

$$C(i,j) = \begin{cases} \min_{i \le k < j} \{C(i,k) + C(k+1,j) + d_{i-1} \times d_k \times d_j\} & i < j \\ 0 & i = j \end{cases}$$

Giải thuật

```
ChainMatrixMult_Memo(d[0 .. n]) {
   C[.., ..] = ∞;
   return   MMM(C, d, 1, n);
}

MMM(C[1 .. n, 1 .. n], d[0 .. n], i, j) {
   if (C[i, j] < ∞)
      return   C[i, j];

   if (i == j)
      C[i, j] = 0;
   else
      for (k = i; k < j; k++) {
         t = MMM(C, d, i, k) + MMM(C, d, k + 1, j) + d[i - 1] * d[k] * d[j];
         if (t < C[i, j])
            C[i, j] = t;
      }
   return C[i, j];
}
```

Ví dụ: Bài toán túi xách 0/1

Hệ thức truy hồi của bài toán là:

$$T[i,j] = \begin{cases} max\{T[i-1,j], v_i + T[i-1, j-w_i]\} & j \ge w_i \\ T[i-1,j] & j < w_i \end{cases}$$

Điều kiện đầu được xác định như sau:

$$\begin{cases} T[0,j] = 0 & j \ge 0 \\ T[i,0] = 0 & i \ge 0 \end{cases}$$

Giải thuật

```
Knapsack(T[0 .. n, 0 .. W] , i, j) {
   if (T[i, j] < 0) {
      if (j < w[i])
         tmp = Knapsack(T, i - 1, j);
      else
         tmp = max(Knapsack(T, i - 1, j), v[i] + Knapsack(T, i-1, j – w[i]));
      T[i, j] = tmp;
   }
   return   T[i, j];
}
Knapsack_Memo() { // global variable: w[1 .. n], v[1 .. n]
   T[,] = -1;
   T[0, 0 .. W] = T[0 .. n, 0] = 0;
   return   Knapsack(T, n, W);
}
```

Nhận xét: Dữ liệu đầu vào là tập gồm 4 đồ vật: $\{w_1 = 2, v_1 = 12đ\}, \{w_2 = 1, v_2 = 10đ\}, \{w_3 = 3, v_3 = 20đ\}, \{w_4 = 2, v_4 = 15đ\}$ và $W = 5$:

	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	12	12	12	12
2	0	–1	12	22	–1	22
3	0	–1	–1	22	–1	32
4	0	–1	–1	–1	–1	**37**