Qui hoạch động

Nguyên lý của sự tối ưu

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

Vấn đề sự chồng lấp trong việc tính toán những thể hiện nhỏ của bài toán.

Ví dụ: Dãy Fibonacci

$$F(n) = F(n-1) + F(n-2) \; với \; F(0) = 0, F(1) = 1$$

Ví dụ: Tính hệ số nhị thức $\binom{n}{k}$ hay $C(n, k)$.

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

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

và tính đệ qui được thể hiện bởi công thức:

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

Công thức tổng quát là:

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

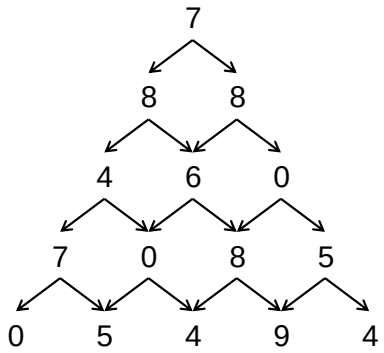
	0	1	2	…	$j-1$	j	…	$k-1$	k
0	1								
1	1	1							
2	1	2	1						
⋮									
$i-1$					$C(i-1, j-1)$	$C(i-1, j)$			
i						$C(i, j)$			
⋮									
k	1								1
⋮									
$n-1$	1							$C(n-1, k-1)$	$C(n-1, k)$
n	1								$C(n, k)$

Giải thuật

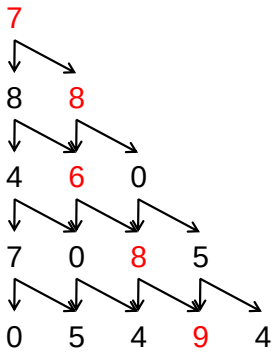
```
Binomial(n, k) {
   for (i = 0; i ≤ n; i++)
      for (j = 0; j ≤ min(i, k); j++)
         if ((j == 0) || (j == i))  C[i][j] = 1;
         else                       C[i][j] = C[i - 1][j - 1] + C[i - 1][j];
   return   C[n][k];
}
```

Bài toán tìm đường

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)

	1	2	3	4	5
1	7				
2	15	15			
3	19	21	15		
4	26	21	29	20	
5	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.

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

$S[0 .. n, 0] = 0$

$S[j, j + 1] = 0$ với $0 \leq j \leq n - 1$.

Phép duyệt tuyến tính trên mảng S tại dòng thứ n sẽ chỉ ra tổng lớn nhất.

Giải thuật

```
S[0 .. n, 0] = S[0 .. n - 1, 1 .. n] = 0;
for (i = 1; i ≤ n; i++)
   for (j = 1; j ≤ i; j++) {
      k = max(S[i - 1][j - 1], S[i-1][j]);
      S[i][j] = a[i][j] + k;
   }
"Tìm và trả về giá trị lớn nhất của dòng thứ n";
```

Mở rộng: Xác định con đường dẫn đến kết quả. Gọi $S[r, c]$ (với $r = n$) chứa kết quả.

Nếu $S[r - 1, c - 1] > S[r - 1, c]$ thì vị trí trước đó là $(r - 1, c - 1)$;

Ngược lại: vị trí trước đó sẽ là $(r - 1, c)$.

Tiến trình này sẽ lùi về cho đến khi đạt tới đỉnh ($r = 1$).

Bài toán đổi tiền xu

Gọi mệnh giá các đồng xu là $\{x_1, x_2, \ldots, x_k\}$ và cho rằng đồng xu có mệnh giá nhỏ nhất là 1 xu.

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

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

Giải thuật

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

   for (cents = 1; cents ≤ money; cents++) {
      int minCoins = cents;
      for (i = 0; i < k; i++) {
         if (coins[i] > cents)
            continue;
         if (C[cents - coins[i]] + 1 < minCoins) {
            minCoins = C[cents - coins[i]] + 1;
            usedCoin = coins[i];
         }
      }
      C[cents] = minCoins;
      lastCoin[cents] = usedCoin;
   }
   return <C[money], lastCoin[]>;
}
```

Mở rộng

Mảng `lastCoin[0..money]` mỗi khi cập nhật giá trị bằng câu lệnh

```
lastCoin[cents] = usedCoin;
```

cho thấy, đồng xu `usedCoin` vừa mới được sử dụng để đổi lượng tiền `cents` xu.

Tìm dãy con tăng nghiêm ngặt dài nhất

Phát biểu: Cho dãy số nguyên dương a_1, a_2, …, 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 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 có giá trị lớn nhất*.

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

với $L(1) = 1$

Giải thuật (đệ qui)

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

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

   if (max < tmpMax)
      max = tmpMax;
   return   tmpMax;
}
lis(a, n, max);
```

Đánh giá: Chi phí thuộc về nhóm $\Theta(2^n)$.

Tiếp cận qui hoạch động

Gọi $Tab[i]$ 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:

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

Giải thuật

```
lis_DP(a[1..n]) {
   tab[1 .. n] = 1;

   for (i = 2; i ≤ n; i++)
      for (j = 1; j < i; j++)
         if ((a[j] < a[i]) && (tab[j] + 1 > tab[i]))
            tab[i] = tab[j] + 1;

   max = "Tìm phần tử lớn nhất trên mảng Tab";
   return   max;
}
cout << lis_DP(a);
```

Đánh giá: Hai vòng lặp `for` lồng nhau cho thấy chi phí của giải thuật là $\Theta(n^2)$.

Mở rộng: Chỉ ra chính xác dãy con tăng tuyệt đối dài nhất. Quá trình dò vết này sẽ dựa vào bảng *Tab*.

Gọi i là vị trí mà tại đó, $Tab[i]$ cho giá trị lớn nhất. Phần tử đứng ngay bên trái của nó trong dãy con tăng tuyệt đối dài nhất phải là phần tử $Tab[j]$ nào đó, sao cho $1 \le j < i$ và thỏa hai điều kiện: $a_j < a_i$ và $Tab[j] + 1 = Tab[i]$.

Khi xác định được phần tử $Tab[j]$, ta cập nhật giá trị $i = j$ và quá trình trên lại tiếp tục tiến sang trái cho đến đầu mảng.

Lập lịch theo quãng thời gian có trọng số

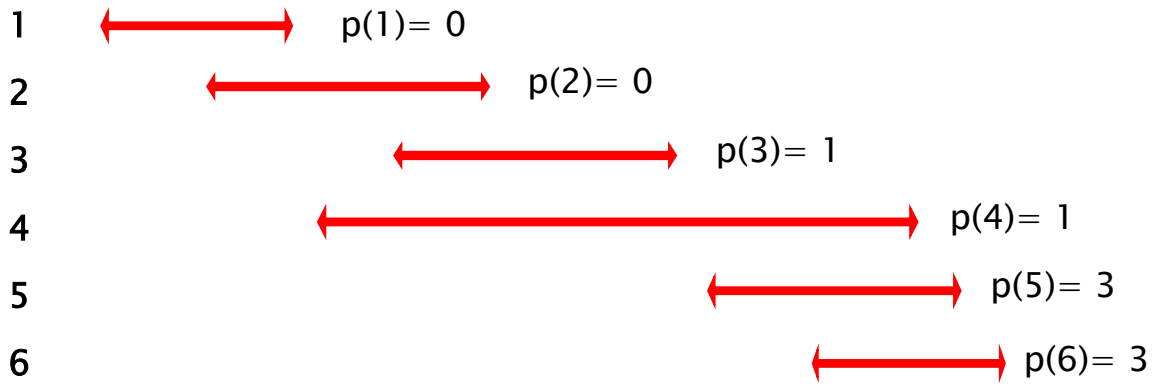
Phát biểu: Cho tập các công việc $J = \{j_1, j_2, \ldots, j_n\}$. Đối với mỗi j_i, thời điểm bắt đầu và kết thúc công việc là $<s_i, f_i>$ và giá trị của công việc này là v_i. Tìm tập con $S = \{j_{s_1}, j_{s_2}, \ldots, j_{s_m}\}$ $(1 \le m \le n)$ thỏa hai điều kiện:

1. Quãng thời gian của chúng không chồng lấp nhau.
2. Tổng các trọng số $\sum_{t=1}^{m} v_{s_t}$ là lớn nhất.

Sắp xếp các công việc trong J theo thời gian kết thúc tăng dần.

Gọi $p(h) = k$ là chỉ mục lớn nhất thỏa (i) $k < h$ và (ii) $f_k < s_h$.

Ví dụ:



Giải thuật

```
p[1 .. n] = 0;
for (h = 2; h ≤ n; h++)
   for (k = h - 1; k > 0; k--)
      if (f[k] < s[h]) {
         p[h] = k;
         break;
      }
```

Gọi $T(h)$ là tổng trọng số lớn nhất đối với dãy công việc từ 1 đến h. Hệ thức truy hồi là:

$$T(h) = max\{v_h + T(p(h)), T(h-1)\}$$

với $T(0) = 0$.

Giải thuật

```
Scheduling(h) {
   if (h == 0)
      return   0;
   return max(v[h] + Scheduling(p[h]), Scheduling(h - 1));
}
```

Tiếp cận qui hoạch động

Gọi T là mảng một chiều có n phần tử, sao cho $T[h]$ chứa tổng trọng số lớn nhất có được đối với dãy công việc từ 1 đến h.

$$T[h] = \begin{cases} max(v[h] + T[p[h]], T[h-1]) & h > 0 \\ 0 & h = 0 \end{cases}$$

Giải thuật

```
Scheduling(v[1 .. n], p[1 .. n]) {
   T[0 .. n] = 0;
   for (h = 1; h ≤ n; h++)
      T[h] = max{v[h] + T[p[h]], T[h - 1]};
   return   T[n];
}

Preprocess(J[1 .. n], s[1 .. n], f[1 .. n], v[1 .. n]) {
   p[1 .. n] = 0;
   sort(J, s, f, v);
   for (h = 2; h ≤ n; h++)
      for (k = h – 1; k > 0; k--)
         if (f[k] < s[h]) {
            p[h] = k;
            break;
         }
   return Scheduling(v, p);
}
```

Mở rộng: Xác định dãy công việc cụ thể.

Như đã biết, công việc h sẽ thuộc về lời giải nếu $v_h + T(p(h)) \geq T(h-1)$. Xét từ công việc cuối cùng j_n:

Nếu $v_n + T(p_n) \geq T(n-1)$ thì j_n thuộc về lời giải. In kết quả thành phần và xét tiếp công việc $j_{p(n)}$;

Ngược lại, bỏ qua j_n và xét công việc j_{n-1}.

Quá trình này đi ngược về đầu mảng T.

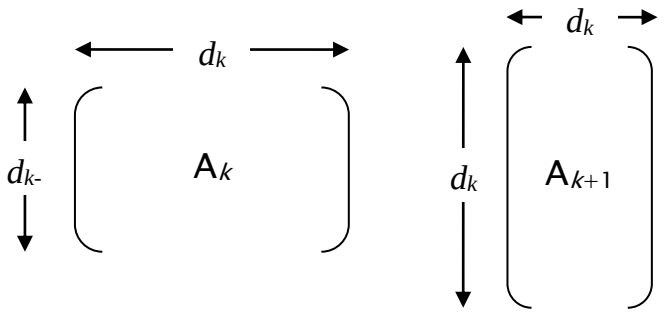
Nhân dãy ma trận

Phát biểu: Xác định thứ tự tối ưu để nhân dãy n ma trận $A_1 \times A_2 \times \ldots \times A_n$ có kích thước $d_0 \times d_1$, $d_1 \times d_2$, …, $d_{n-1} \times d_n$ với chi phí thấp nhất.
Chú ý: Tích hai ma trận kích thước $p \times q$ và $q \times r$ đòi hỏi $p \times q \times r$ phép nhân và tạo ra ma trận kích thước $p \times r$.

Giả sử cần nhân 4 ma trận A × B × C × D với kích thước lần lượt là 50×20, 20×1, 1×10 và 10×100. So sánh *ba* trong số *năm* thứ tự nhân 4 ma trận nêu trên:

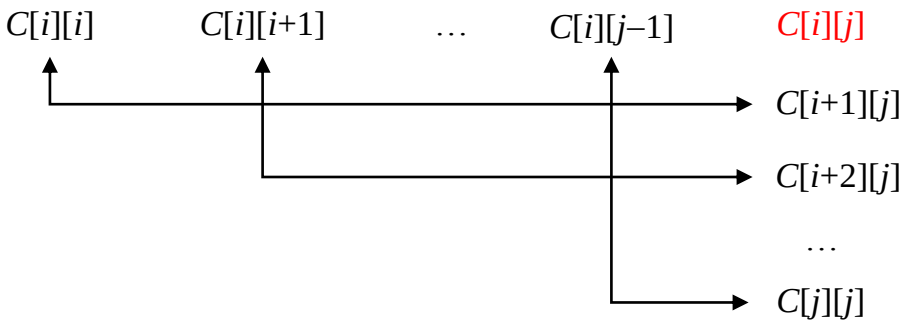
Thứ tự nhân	Số phép tính	
A × ((B × C) × D)	20×1×10 + 20×10×100 + 50×20×100 =	120200
(A × (B × C)) × D	20×1×10 + 50×20×10 + 50×10×100 =	60200
(A × B) × (C × D)	50×20×1 + 1×10×100 + 50×1×100 =	7000

Qui ước: Nếu $A_k \times A_{k+1}$ $(1 \le k < n)$ thì kích thước của ma trận A_k và A_{k+1} lần lượt là $d_{k-1} \times d_k$ và $d_k \times d_{k+1}$. Vậy, d_0 sẽ là số dòng của ma trận A_1 và d_n là số cột của ma trận A_n.



Xét tích $A_i \times A_{i+1} \times \ldots \times A_j$ 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]) {
   for (i = 1; i ≤ n; i++)
      C[i][i] = 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];
}
```

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 khả năng dùng các khóa tìm kiếm là như nhau. Nếu có thông tin về tần suất sử dụng khóa, để tăng hiệu quả truy xuất, chúng ta:

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

Qui ước: Gọi

- $key_1, key_2, \ldots, key_n$ là khóa của n nút trên cây. Cho rằng $key_1 < key_2 < \ldots < key_n$.
- p_i là xác suất để key_i trở thành khóa tìm kiếm
- c_i là số phép so sánh để tìm ra key_i:

$$c_i = level(key_i) + 1$$

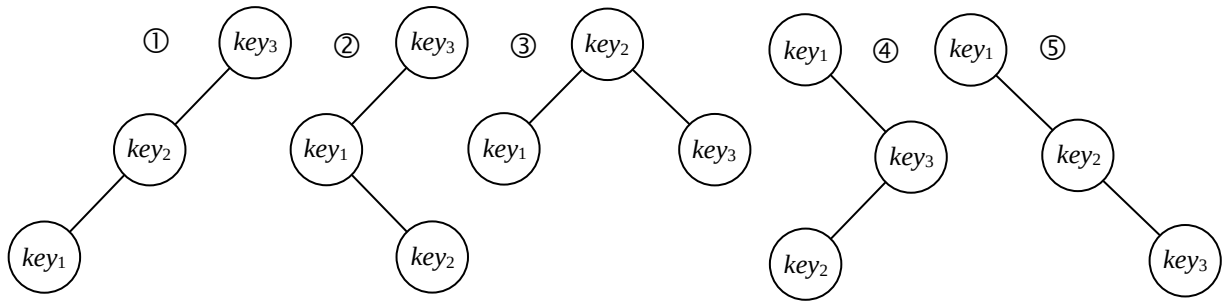
- Chi phí chung để tìm một khóa bất kỳ trên cây là

$$Cost = \sum_{i=1}^{n} c_i \times p_i$$

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 xác suất chọn key_1, key_2, key_3 làm khóa tìm kiếm lần lượt là $p_1 = 0.7$, $p_2 = 0.2$, $p_3 = 0.1$. Cho rằng $key_1 < key_2 < key_3$.

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 T_i^j với $1 \le i \le j \le n$ là cây nhị phân tìm kiếm chứa các khóa từ key_i đến key_j.

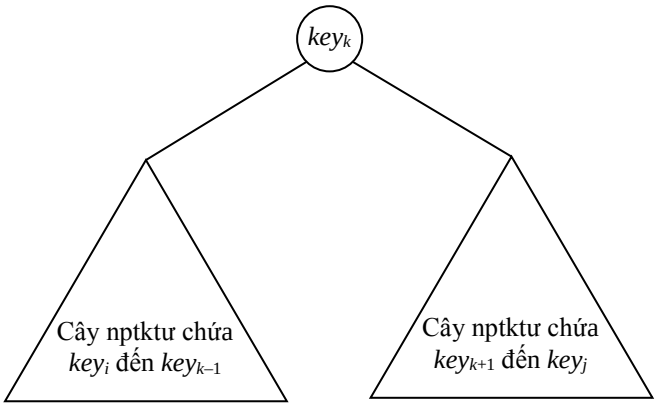
Gọi $C(i, j)$ là chi phí chung nhỏ nhất để tìm một khóa bất kỳ trên cây nhị phân tìm kiếm chứa các khóa từ key_i đến key_j.

Nếu $i = j$: Khi đó, 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.



$$C(i,k-1) = \sum_{s=i}^{k-1} c_s \times p_s \text{ và } C(k+1,j) = \sum_{s=k+1}^{j} c_s \times p_s$$

Đối với cây có nút gốc mang khóa key_k:

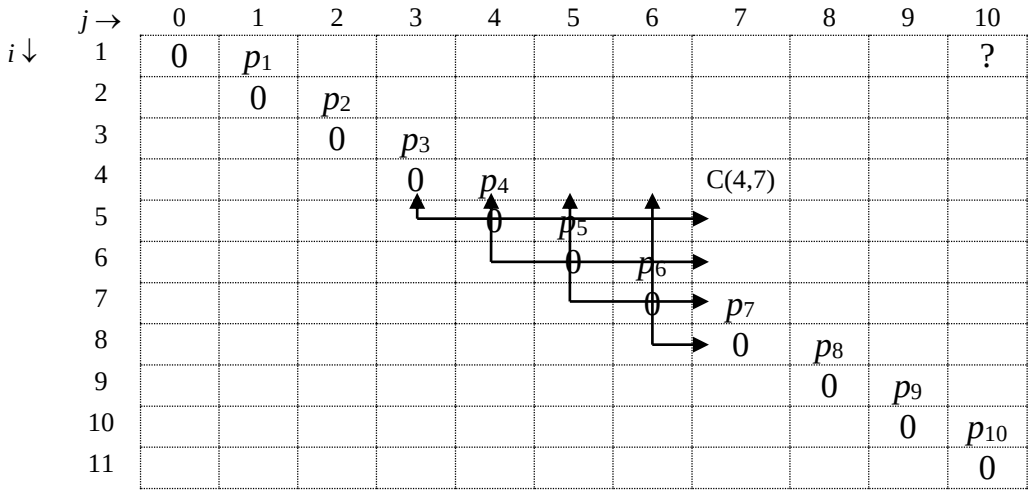
$$\sum_{s=i}^{k-1} (c_s + 1) \times p_s = C(i,k-1) + \sum_{s=i}^{k-1} p_s \text{ và } \sum_{s=k+1}^{j} (c_s + 1) \times p_s = C(k+1,j) + \sum_{s=k+1}^{j} p_s$$

Chi phí tìm kiếm khóa key_k tại nút *gốc*: p_k

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

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

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



Giải thuật

```
OptimalBST(p[1..n]) {
   C[1 .. n + 1][0 .. n], R[1 .. n + 1][0 .. n];
   for (i = 1; i ≤ n; i++) {
      C[i][i - 1] = 0;
      C[i][i]     = p[i];
      R[i][i]     = i;
   }
   C[n + 1][n] = 0;
   for (diag = 1; diag < n; diag++)
      for (i = 1; i ≤ n – diag; i++) {
         j = i + diag;
         minval = min_{i ≤ k ≤ j}(C[i][k - 1] + C[k + 1][j]);
         R[i][j] = Giá trị k tìm được;
         C[i][j] = minval + (p[i] + p[i+1] + … + p[j]);
      }
   return      C[1][n], R;
}
```

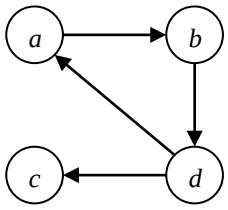
Mở rộng

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

Giải thuật (của) Warshall tìm bao đóng chuyển tiếp

Định nghĩa: Bao đóng chuyển tiếp của đồ thị có hướng n đỉnh được chỉ ra như là ma trận $\mathcal{T}$ kích thước $n \times n$, trong đó phần tử $\mathcal{T}[i, j]$ tại dòng i, cột j ($1 \le i, j \le n$) là *đúng* nếu tồn tại một con đường có chiều dài dương từ đỉnh nhãn i đến đỉnh nhãn j, nếu không, $\mathcal{T}[i, j]$ = *sai*.

Ví dụ:



(a)

$A =$

	a	b	c	d
a	0	1	0	0
b	0	0	0	1
c	0	0	0	0
d	1	0	1	0

(b)

$T =$

	a	b	c	d
a	1	1	1	1
b	1	1	1	1
c	0	0	0	0
d	1	1	1	1

(c)

Ý tưởng: Xây dựng bao đóng chuyển tiếp của đồ thị có hướng n đỉnh thông qua dãy $n + 1$ ma trận luận lý:

$$R^{(0)}, R^{(1)}, \ldots, R^{(k-1)}, R^{(k)}, \ldots, R^{(n)}.$$

Phần tử $r_{ij}^{(k)}$ tại dòng i, cột j của ma trận $R^{(k)}$ ($k \in [0, n]$) bằng *đúng* nếu và chỉ nếu tồn tại một con đường có hướng (chiều dài dương) đi từ đỉnh nhãn i đến đỉnh nhãn j mà mỗi đỉnh trung gian, nếu tồn tại, phải có nhãn từ k trở xuống.

Công thức xây dựng $R^{(k)}$:

$$r_{ij}^{(k)} = \begin{cases} \text{đúng} & \left(r_{ij}^{(k-1)} = \text{đúng}\right) \cup \left(r_{ik}^{(k-1)} = \text{đúng và } r_{kj}^{(k-1)} = \text{đúng}\right) \\ sai & \text{ngược lại} \end{cases}$$

Giải thuật

```
Warshall(A[1 .. n][1 .. n]) {
   R(0) = A;
   for (k = 1; k ≤ n; k++)
      for (i = 1; i ≤ n; i++)
         for (j = 1; j ≤ n; j++)
            R(k)[i][j] = R(k-1)[i][j] || (R(k-1)[i][k] && R(k-1)[k][j]);
   return   R(n);
}
```

Cải tiến 1

Giải thuật

```
Warshall(A[1 .. n][1 .. n]) {
   for (k = 1; k ≤ n; k++)
      for (i = 1; i ≤ n; i++)
         for (j = 1; j ≤ n; j++)
            A[i][j] = A[i][j] || (A[i][k] && A[k][j]);
   return   A;
}
```

Cải tiến 2

Giải thuật

```
Warshall(A[1 .. n][1 .. n]) {
   for (k = 1; k ≤ n; k++)
      for (i = 1; i ≤ n; i++)
         if (A[i][k])
            for (j = 1; j ≤ n; j++)
               if (A[k][j])
                  A[i][j] = 1;
   return   A;
}
```

Tổng các tập con

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

Dùng bảng V kích thước $(n + 1)\times(k + 1)$ để lưu giá trị. Nhận thấy:

- Nếu tồn tại *tập con của tập* $\{a_1, \ldots, a_i\}$ có tổng là j $(1 \le j \le k)$: $V(i, j) = 1$
- Ngược lại: $V(i, j) = 0$.

$$V(i,j) = \begin{cases} 1 & V(i-1,j) = 1 \text{ hoặc } V(i-1,j-a_i) = 1 \\ 0 & \text{ngược lại} \end{cases}$$

với $V(0, 1 .. k) = 0$ và $V(0 .. n, 0) = 1$.

Sau khi làm đầy bảng, nếu $V(n, k) = 1$ thì tồn tại một tập con của tập $\{a_1, \ldots, a_n\}$ có tổng là k, ngược lại thì không.

Giải thuật

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

Để in ra một tập con kết quả, ta truy ngược từ phần tử V(n, k) với nhận định rằng, một phần tử a_i thuộc về tập kết quả nếu:

$$V(i-1, j-a_i) = 1 \text{ và } V(i-1, j) = 0$$

Mở rộng 1

Phát biểu: Cho tập $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ gồm n số *nguyên dương*. Cho biết, tập này có thể chia thành hai tập con $\mathcal{A}_1$ và $\mathcal{A}_2$ sao cho: $\mathcal{A}_1 \cap \mathcal{A}_2 = \varnothing$, $\mathcal{A}_1 \cup \mathcal{A}_2 = \mathcal{A}$ và tổng của các phần tử của hai tập bằng nhau.

Có thể áp dụng cách giải trên với $k = \frac{1}{2}\sum_{i=1}^{n} a_i$.

Chú ý: Nếu $\sum_{i=1}^{n} a_i$ là số lẻ thì chắc chắn không tồn tại lời giải.

Mở rộng 2

Phát biểu: Cho tập $\mathcal{A} = \{a_1, a_2, \ldots, a_n\}$ gồm n số *nguyên dương*. Tìm cách chia tập này thành hai tập con $\mathcal{A}_1$ và $\mathcal{A}_2$ thỏa những điều kiện sau:

1. $\mathcal{A}_1 \cap \mathcal{A}_2 = \varnothing$, $\mathcal{A}_1 \cup \mathcal{A}_2 = \mathcal{A}$
2. Nếu gọi $S_{\mathcal{A}}$ là tổng của các phần tử của tập $\mathcal{A}$ thì $|S_{\mathcal{A}1} - S_{\mathcal{A}2}|$ là nhỏ nhất.

Gọi $T = \frac{1}{2}\sum_{i=1}^{n} a_i$.

- Nếu T là số nguyên: Kiểm tra liệu giá trị của $V(n, T) = 1$ hay không? Nếu đúng thì có ngay lời giải:
$$|S_{\mathcal{A}1} - S_{\mathcal{A}2}| = 0.$$
- Nếu (*i*) T là số lẻ hoặc (*ii*) T là nguyên nhưng $V(n, T) = 0$: $\forall j$ sao cho $V(n, j) = 1$, luôn tồn tại hai tập con tách rời $\mathcal{A}_1$ và $\mathcal{A}_2$ với $S_{\mathcal{A}1} = j$ và $S_{\mathcal{A}1} = 2T - j$. Vậy:
$$|S_{\mathcal{A}1} - S_{\mathcal{A}2}| = |j - (2T - j)| = |2j - 2T| = |2T - 2j|$$
Để tối thiểu trị tuyệt đối này, ta tìm: $min\{(T - j)$ với $V(n, j) = 1$ và $j \le T\}$

Bài toán ba lô

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

Gọi $V(i, j)$ 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 ($w_1, w_2, \ldots, w_i$) và cho được vào ba lô có sức chứa j.

Hệ thức truy hồi là:

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

Điều kiện đầu được xác định như sau:

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

Cuối cùng, $V(n, W)$ là giá trị của tập con đáng giá nhất trong số n đồ vật chứa được trong ba lô có sức chứa W.

Giải thuật

```
Knapsack(w[1 .. n], v[1 .. n], W) {
   float V[0 .. n][0 .. W];

   V[0 .. n][0] = V[0][1 .. W] = 0;

   for (i = 0; i ≤ n; i++)
      for (j = 1; j ≤ W; j++)
         if (j ≥ w[i])
            V[i][j] = max{V[i - 1][j], V[i - 1][j – w[i]] + v[i]};
         else
            V[i][j] = V[i - 1][j];

   return   V[n][W];
}
```

Mở rộng: Dò vết ngược trên bảng, xuất phát từ $V(n, W)$ và đi dần về dòng đầu tiên.

Bài toán đường đi người bán hàng

Cho đồ thị liên thông (có hướng, có trọng số) $G = (V, E)$ với tập các đỉnh là $V = \{v_1, v_2, \ldots, v_n\}$. Hai đỉnh không kề nhau có trọng số là $+\infty$.

Không mất đi tính tổng quát, gọi v_1 là đỉnh bắt đầu của mọi chu trình.

Ý tưởng: "Nếu v_k 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_k quay về v_1, sẽ phải là con đường ngắn nhất từ v_k về v_1 đi qua mọi đỉnh còn lại đúng một lần".

Gọi:

- W: Ma trận kề biểu diễn G.
- A: Tập các đỉnh ($\subseteq V$).
- $D[v_i][A]$: 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, $\forall i \in [2, n], v_i \notin A$:

$$D[v_i][A] = \begin{cases} \min\limits_{j:v_j \in A} \left(W[i][j] + D[v_j][A - \{v_j\}]\right) & A \neq \emptyset \\ W[i][1] & A = \emptyset \end{cases}$$

Giải thuật

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

   for (i = 2; i ≤ n; i++)
      D[i][∅] = W[i][1];

   for (k = 1; k ≤ n - 2; k++)       // Kích thước của tập A
      for (mọi tập con A (⊆ V \ {v1}) chứa k đỉnh)
         for (i: i ≠ 1 và vi ∉ A) {
            D[i][A] = min_{j:vj∈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}];
}
```

Xây dựng chu trình tối ưu

Đầu tiên, gọi $A = V - \{v_1\}$. Xét giá trị của P[1][A] và cho rằng nó bằng với k. Điều này thể hiện hai đỉnh đầu tiên của chu trình tối ưu là $v_1 \rightarrow v_k$.

Tập A sẽ được cập nhật lại: $A = A \setminus \{v_k\}$. Tiếp tục, xét giá trị của P[k][A] và cho rằng nó bằng với h. Điều này thể hiện ba đỉnh đầu tiên của chu trình tối ưu là $v_1 \rightarrow v_k \rightarrow v_h$.

Tiếp tục cập nhật tập A và xử lý tương tự cho đến khi A bằng rỗng thì dừng.