动态规划

陈长建

计算机科学系

例2 - 矩阵连乘问题

给定n个矩阵{A_1， A_2，…, A_n},其中A_i与A_{i+1}是可乘的。考察这N个矩阵的连乘积 $A_1A_2...A_n$

分析：

由于矩阵乘法满足结合律，故计算矩连乘积 $A_1A_2...A_n$ 可以有多个计算次序。

例如：
- $(A_1(A_2(A_3A_4)))$ 等等
- $(A_1((A_2A_3)A_4))$
- $((A_1A_2)(A_3A_4))$

小练习：合并果子-1

在一个果园里，果农已经将所有的果子打了下来，而且按圆形区域堆放了若干堆。最后果农要把所有的果子合并成一堆。

每一次合并，果农总是把两堆相邻果子合并到一起，消耗的体力等于两堆果子的重量之和。可以看出，所有的果子经过n-1次合并之后，就只剩下一堆了。果农在合并果子时总共消耗的体力等于每次合并所耗体力之和。

假定每个果子重量都为1，并且已知果子堆数和每堆的数目，你的任务是设计出合并的次序方案，使得果农耗费的体力最少，并输出这个最小的体力耗费值。

本章要点:

- 理解动态规划算法的概念
- 掌握动态规划算法的基本要素
- 掌握设计动态规划算法的步骤
- 通过范例学习动态规划算法设计策略
  - 最大子段和、最长公共子序列、矩阵链连乘、凸多边形最优三角剖分、0-1背包问题

例3-最长公共子序列

定义 一个给定序列的子序列是在该序列中删除若干元素后得到的序列。即

若给定序列 X={x1,x2,...,xm}，则另一序Z={z1,...,zk}是X的子序列是指:

存在一个严格递增下标序列{i1,i2,...,ik}使得对于所有j=1,2,...,k有：$Z_j=X_{ij}$

例如 序列 X={A，B，C，B，D，A，B}

子序列 Z={B，C，D，B}

相应的递增下标序列为{2，3，5，7}。

最长公共子序列

公共子序列

给定2个序列 X 和 Y，当另一序列 Z 既是X的子序列又是Y的子序列时，称Z是序列X和Y的公共子序列。

最长公共子序列问题

给定2个序列 X={x1,x2,...,xm}和 Y={y1,y2,...,yn}，找出 X 和 Y 的最长公共子序列。

例如：字符串13455和245576的最长公共子序列为455

字符串acdfg和adfc的最长公共子序列为adf

最长公共子序列

LCS（Longest Common Subsequence）的应用

- 求两个序列中最长的公共子序列算法，广泛的应用在图形相似出路、媒体流的相似比较、计算生物学方面。生物学家常常利用该算法进行基因序列比对，由此推测序列的结构、功能和演化过程。
- LCS可以描述两段文字之间的“相似度”，即它们的雷同程度，从而能够用来辨别抄袭。另一方面，对一段文字进行修改之后，计算改动前后文字的最长公共子序列，将除此子序列外的部分提取出来，这种方法判断修改的部分，往往十分准确。简而言之，百度知道，百度百科都用得上。

最长公共子序列的结构

求最长公共子序列问题，最容易想到的算法----枚举法

即，对X的所有子序列，检查它是否也是Y的子序列，从中找出最长的子序列。

但 X 共有2^M个不同的子序列。枚举法 $O(2^m)$

事实上，

最长公共子序列问题，具有最优子结构性质。

最优子结构性质

设序列$X_m=\{x_1,x_2,\ldots,x_m\}$和$Y_n=\{y_1,y_2,\ldots,y_n\}$的最长公共子序列为$Z_k=\{z_1, z_2,\ldots, z_k\}$，则

- 若$x_m=y_n$ 则 $z_k=x_m=y_n$ 且 Z_{k-1}是X_{m-1}和 Y_{n-1}的最长公共子序列。
- 若$x_m\neq y_n$ 且 $z_k\neq x_m$ 则 Z是X_{m-1}和 Y 的最长公共子序列
- 若$x_m\neq y_n$ 且 $z_k\neq y_n$ 则 Z是X_m和 Y_{n-1} 的最长公共子序列

由此可见，2个序列的最长公共子序列包含了这2个序列的前缀的最长公共子序列。因此，最长公共子序列问题具有最优子结构性质。

子问题的递归结构

由最长公共子序列问题的最优子结构性质建立子问题最优值的递归关系。

c[i][j]: 记录序列 X_i和Y_j的最长公共子序列的长度。其中，

$X_i=\{x_1,x_2,\ldots,x_i\}$　　$Y_j=\{y_1,y_2,\ldots,y_j\}$

由最优子结构性质可建立递归关系如下：

$$c[i][j]=\begin{cases}0 & i=0,j=0\\ c[i-1][j-1]+1 & i,j>0;x_i=y_j\\ \max\{c[i][j-1],c[i-1][j]\} & i,j>0;x_i\neq y_j\end{cases}$$

计算最优值

由于在所考虑的子问题空间中，总共有θ(mn)个不同的子问题，因此，用动态规划算法自底向上的计算最优值能提高算法的效率

```
void LCSLength(int m，int n，char *x，char *y，int **c，int **b)
{ int i，j;
    for (i = 1; i <= m; i++)  c[i][0] = 0;
    for (i = 1; i <= n; i++)   c[0][i] = 0;
    for (i = 1; i <= m; i++)
      for (j = 1; j <= n; j++)
      {  if ( x[i]==y[j] )              {  c[i][j]=c[i-1][j-1]+1;   b[i][j]="↖";   }
        else
          if (c[i-1][j]>=c[i][j-1])   {  c[i][j]=c[i-1][j];          b[i][j]="↑";    }
          else
                                          {   c[i][j]=c[i][j-1];          b[i][j]="←";   }
}     }
```

O(mn)

构造最长公共子序列

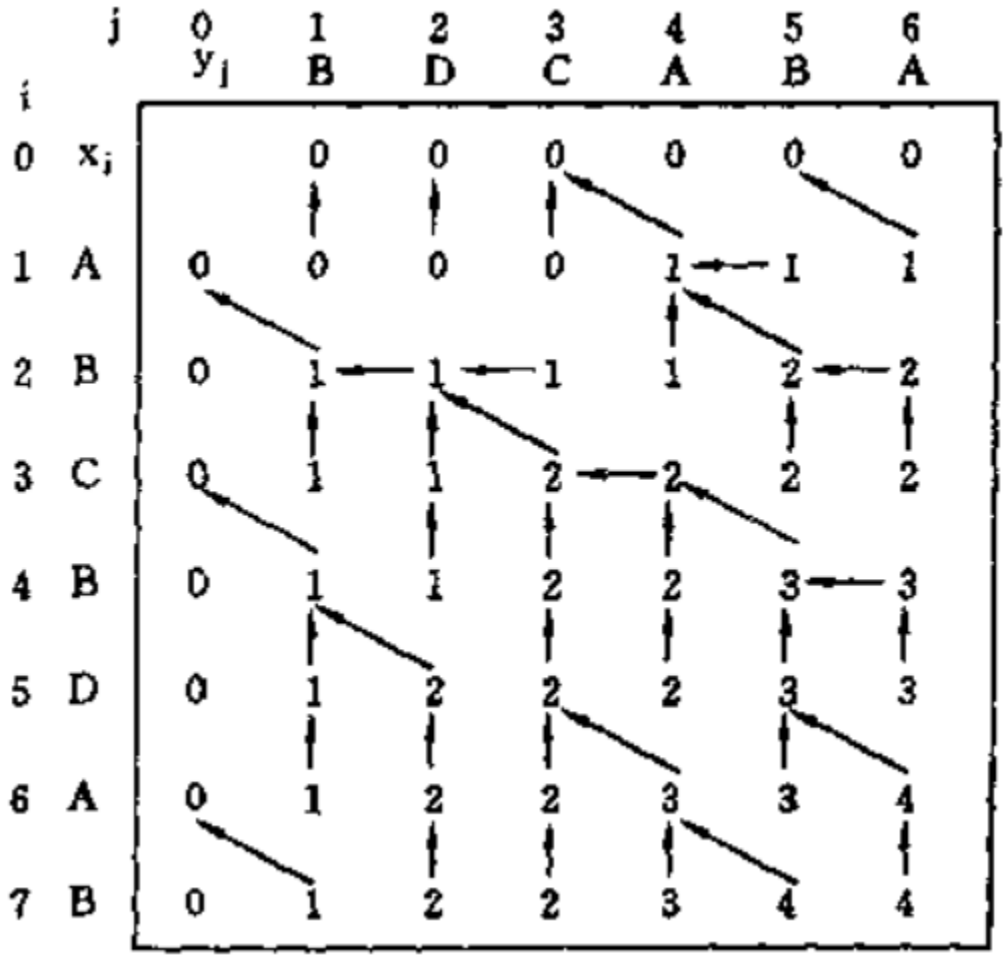
```
void  LCS(int i，  int j，  char *x，  int **b)
   {
     if (i ==0 || j==0) return;
     if  (b[i][j]== “↖”)
                    {    LCS(i-1，  j-1，  x，  b);
                          cout<<x[i];          }
    else
      if (b[i][j]== “↑”)
            LCS(i-1，  j，  x，  b);
      else
            LCS(i，  j-1，  x，  b);
}
```

O(m+n)

练习

- ABCBDAB
- BDCABA

结果



j		0	1	2	3	4	5	6
		y_j	B	D	C	A	B	A
i								
0	x_i		0	0	0	0	0	0
1	A	0	0	0	0	1	1	1
2	B	0	1	1	1	1	2	2
3	C	0	1	1	2	2	2	2
4	B	0	1	1	2	2	3	3
5	D	0	1	2	2	2	3	3
6	A	0	1	2	2	3	3	4
7	B	0	1	2	2	3	4	4

LCS:<B,C,B,A>

算法的改进

在算法LCSLength和LCS中，可进一步将数组b省去。

事实上，数组元素c[i][j]的值仅由c[i-1][j-1]，c[i-1][j]和c[i][j-1]这3个数组元素的值所确定。

如果只需要计算最长公共子序列的长度，则算法的空间需求可大大减少。事实上，在计算c[i][j]时，只用到数组c的第i行和第i-1行。因此，用2行的数组空间就可以计算出最长公共子序列的长度。进一步的分析还可将空间需求减至O(min(m,n))。

Question?

•题目：给定一个长度为N的数组，找出一个最长的单调递增子序列。

•例如：给定数组{5，6，7，1，2，8}则其最长的单调递增自序列为{5，6，7，8}，长度为4。

•求解：怎么用LCS解决这个问题?

•分析：原数组={5，6，7，1，2，8}

排序后={1，2，5，6，7，8}

例4 – 最大子段和

给定由N个整数（可能有负整数）组成的序列$a_1,a_2,\ldots,a_n$,求该序列形如 $a_i+a_{i+1}+\ldots+a_j$的子段和的最大值。

当所有整数均为负整数时，定义其最大子段和为0

例如：

当$\{a_1,a_2,\ldots,a_6\}=\{-1，11，-4，13，-5，-2\}$时

其最大子段和为 20

算法

算法1：对所有的(i,j)对，顺序求和$a_i+\ldots+a_j$并比较出最大的和

算法2：分治策略，将数组分成左右两半，分别计算左边的最大和、右边的最大和、跨边界的最大和，然后比较其中最大者

算法3：动态规划

算法1 最大子段和问题的简单算法

思路如下：

以a_0开始: $\{a_0\},\{a_0,a_1\},\{a_0,a_1,a_2\}.....\{a_0,a_1,.....a_n\}$，共n个

以a_1开始: $\{a_1\}$, $\{a_1,a_2\},\{a_1,a_2,a_3\}.....\{a_1,a_2,.....a_n\}$共n-1个

.....

以a_n开始:$\{a_n\}$共1个

一共(n+1)*n/2个连续子段，使用枚举，那么应该可以得到以下算法：

算法1 最大子段和问题的简单算法

```
int  MaxSum(int *a,  int n,   int *besti ,  int *bestj)
{    int  sum=0;
     for( i=1; i<=n;  i++)
          for( j=i; j<=n;  j++)
          {   T=0;
              for( k=i; k<=j;  k++)  T+=a[k];
               if (T>sum) { sum=T, *besti=i;  *betsj=j;     }
     return sum;
}
```

$O(n^3)$

改进后算法

```
int  MaxSum(int *a,  int n,   int *besti ,  int *bestj)
{    int  sum=0;
   for( i=1; i<=n;  i++)
       {   T=0;
          for( j=i; j<=n;  j++)
          {   T+=a[j];
              if ( T>sum) { sum=T, *besti=i;  *betsj=j;     }
  return sum;
}
```

$O(n^2)$

算法2 最大子段和问题的分治算法

从问题的解的结构可以看出，它适合分治法

将序列 a[1:n] 分为长度相等的两段 a[1: n/2] a[n/2+1: n] 分别求出这两段的最大子段和，则a[1:n] 的最大子段和有三种情形

（1） a[1:n] 的最大子段和与a[1: n/2]的最大子段和相同。

(在前半部分)

（2） a[1:n] 的最大子段和与 a[n/2+1: n]的最大子段和相同。

(在后半部分)

（3） a[1:n] 的最大子段和为 $a_i+a_{i+1}+\ldots+a_j$ (在中间部分)

其中： $1 \le i \le n/2$ $n/2+1 \le j \le n$

情形（1） 和（2） 可递归求得

情形（3）一定 包括元素 a[n/2] 和 a[n/2+1]

因此，

在a[1: n/2]中， 求s1=max $\{a_i+a_{i+1}+\ldots+a_{n/2}\}$ i=1,2,.....,n/2

在a[n/2+1: n]中 求s2=max $\{a_{n/2+1}+\ldots+a_j\}$ j=n/2+1, ...,n

则s1+s2 即为情形（3)时的最大值。

$$T(n)=\begin{cases} O(1) & n<=0 \\ 2T(n/2)+O(n) & n>0 \end{cases} \Rightarrow T(n)=O(n\log n)$$

求最大子段和问题分治算法1

```
int  MaxSubSum(int *a,  int L, int R )
{    int  sum=0;
     if (L==R) sum=(a[L]>0)?a[L]:0;
     else {     int C=(L+R)/2;
                int Lsum= MaxSubSum(a, L, C );
                int Rsum= MaxSubSum(a, C+1, R );
                int s1=0,   Lefts=0;
                for( i=C;  i>L;  - -i )
                {  lefts+=a[i];
                    if(lefts>s1) s1=lefts;
                }
```

求最大子段和问题分治算法2

```
int s2=0,   rights=0;
            for( j=C+1;  j<R;  ++j )
               {   rights +=a[j];
                   if(rights >s2)  s2= rights; }
            sum=s1+s2;
            if (sum<Lsum)  sum=Lsum;
            if (sum<Rsum)  sum=Rsum;
    }
    return sum;
}
```

O(nlogn)

算法3 最大子段和问题的动态规划算法

从对上述分治算法的分析可看出

若记

$$b[j] = \text{MAX}\{\ a[i]+a[i+1]+\ldots\ldots+a[j]\ \} \quad i=1, 2,\ldots., j$$

则最大子段和为

$$\text{MAX}\{\ b[1]\ ,\ b[2],\ b[3]\ ,\ldots\ldots,\ b[n]\ \}$$

如何求b[j]?

$$b[j]=\begin{cases} b[j-1]+a[j] & \text{当 } b[j-1]>0 \text{ 时} \\ a[j] & \text{当 } b[j-1]<0 \text{ 时} \end{cases}$$

$j=1, 2,\ldots., n$

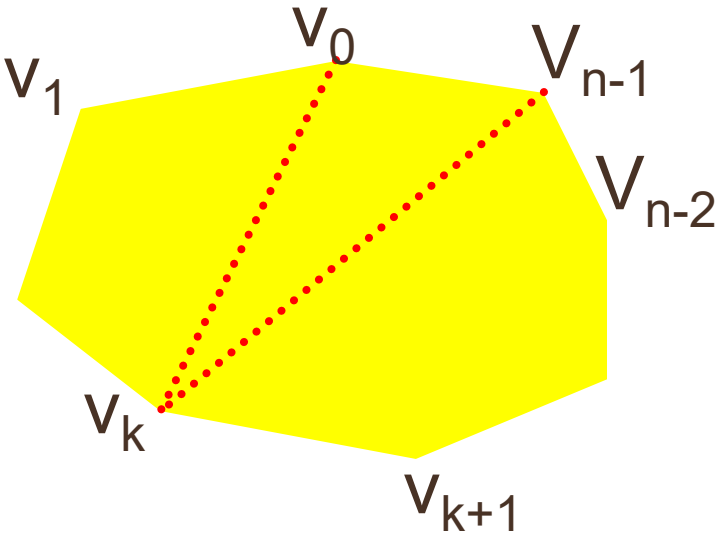
求最大子段和问题动态规划算法

```
int  MaxSum(int *a,  int n){
     int  sum=0，  b=0;
     for( j=1; j<=n;  j++){
          if ( b>0)   b+=a[j];
           else   b=a[j];
            if(b>sum) sum= b;
       }
     return sum;
}
```

例5 – 凸多边形最优三角剖分

- **凸多边形：** 用多边形顶点的逆时针序列表示凸多边形，即$P=\{v_0,v_1,\ldots,v_{n-1}\}$表示具有n条边的凸多边形。

- **弦：** 若v_i与v_j是多边形上不相邻的2个顶点，则线段v_iv_j称为多边形的一条弦。弦将多边形分割成2个多边形$\{v_i,v_{i+1},\ldots,v_j\}$和$\{v_j,v_{j+1},\ldots v_i\}$。

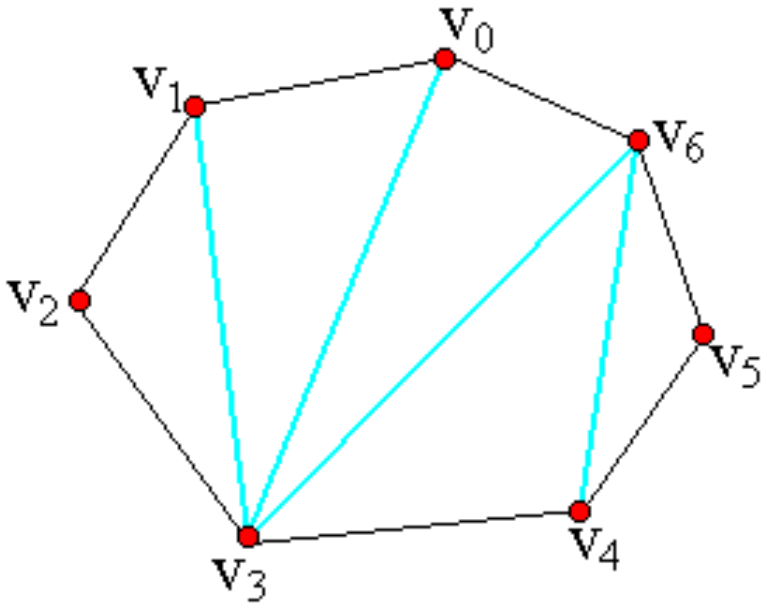


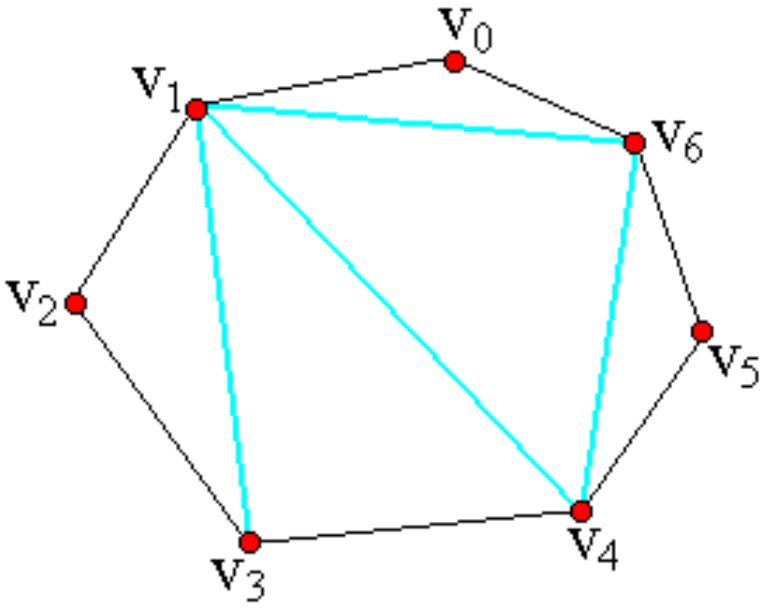
概念

- **多边形的三角剖分：**将多边形分割成互不相交的三角形的弦的集合T。

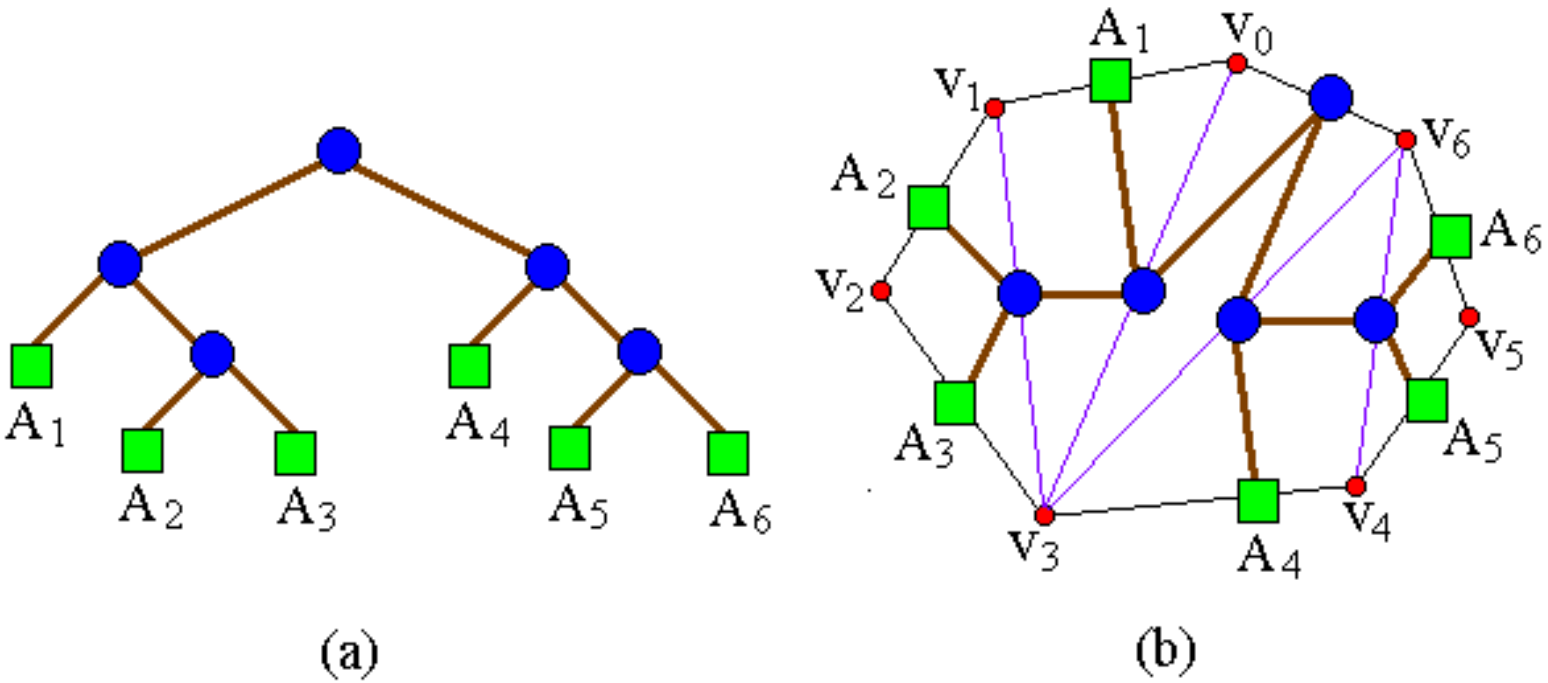
- **凸多边形最优三角剖分：**给定凸多边形P，以及定义在由多边形的边和弦组成的三角形上的权函数w。要求确定该凸多边形的三角剖分，使得该三角剖分中诸三角形上权之和为最小。

凸多边形三角剖分举例





三角剖分的结构及其相关问题

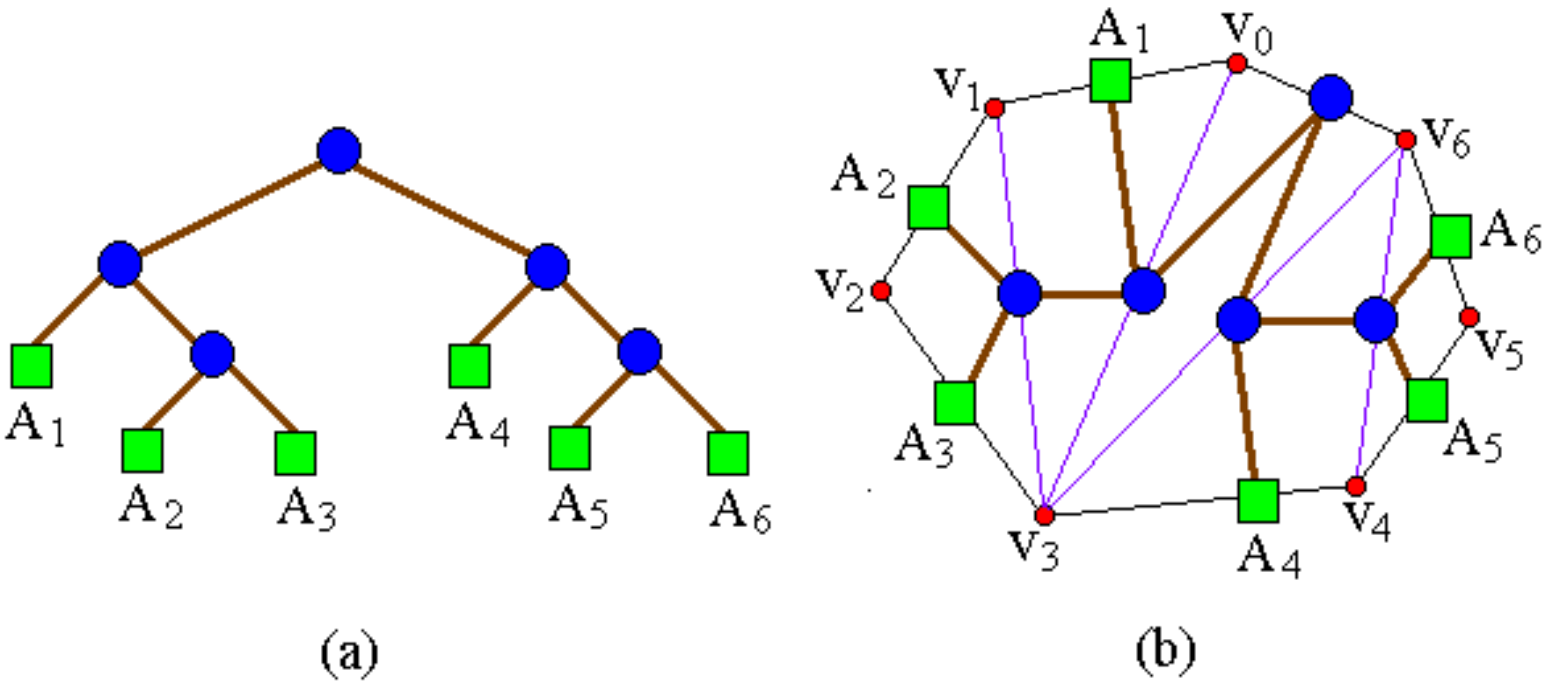


一个表达式的完全加括号方式相应于一棵完全二叉树，称为表达式的语法树。

例如，完全加括号的矩阵连乘积$((A_1(A_2A_3))(A_4(A_5A_6)))$所相应的语法树如图 (a)所示。

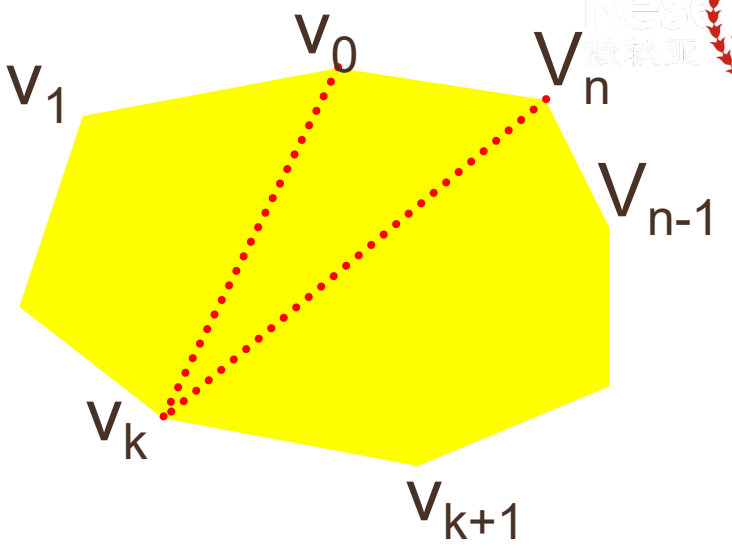
凸多边形$\{v_0,v_1,\ldots v_{n-1}\}$的三角剖分也可以用语法树表示。例如，图(b)中凸多边形的三角剖分可用图 (a)所示的语法树表示。

三角剖分的结构及其相关问题



矩阵连乘积中每个矩阵A_i对应于凸(n+1)边形中的一条边$v_{i-1}v_i$。
三角剖分中的一条弦v_iv_j，i<j，对应于矩阵连乘积A[i+1:j]。

最优子结构性质



- **凸多边形的最优三角剖分问题有最优子结构性质。**

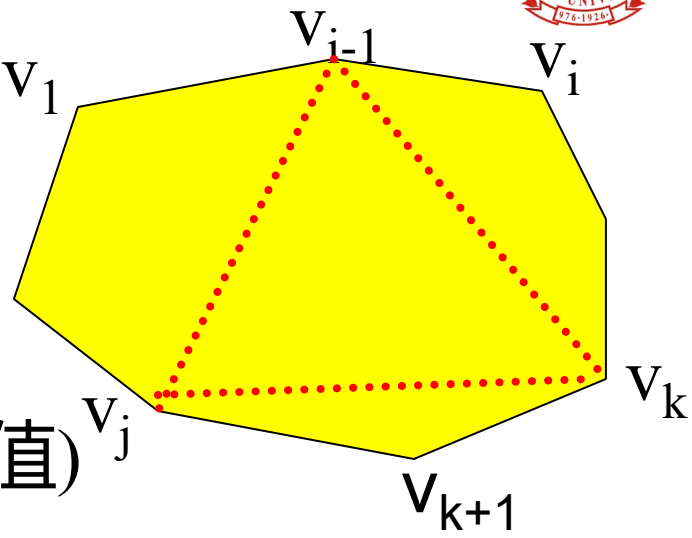
- **理由：**若凸(n+1)边形$P=\{v_0,v_1,\ldots,v_n\}$的最优三角剖分T包含三角形$v_0v_kv_n$，$1\le k\le n-1$，则T的权为3个部分权的和：**三角形$v_0v_kv_n$**的权，子多边形**$\{v_0,v_1,\ldots,v_k\}$**和**$\{v_k,v_{k+1},\ldots,v_n\}$**的权之和。
- 可以断言，由T所确定的这2个子多边形的三角剖分也是最优的。
- 因为若有$\{v_0,v_1,\ldots,v_k\}$或$\{v_k,v_{k+1},\ldots,v_n\}$的更小权的三角剖分将导致T不是最优三角剖分的矛盾。

最优三角剖分的递归结构

- 定义t[i][j]，$1 \le i < j \le n$为凸子多边形$\{v_{i-1}, v_i, \ldots, v_j\}$的最优三角剖分所对应的权函数值，即其最优值。
- **约定：两个顶点的退化多边形$\{v_{i-1}, v_i\}$具有权值0。**
- **问题转化为：计算的凸(n+1)边形P的最优权值为t[1][n]。**

递归关系

当j-i≥1时，凸子多边形{v_{i-1}, v_i,…,v_j}至少有3个顶点。对i≤k≤j-1。



t[i][j]=t[i][k]+t[k+1][j]+($\Delta v_{i-1}v_kv_j$的权值)

k的确切位置待定，但k的所有可能位置只有j-i个，从中可选出使t[i][j]值达到最小的位置。

$$t[i][j]=\begin{cases}0 & i=j\\ \min\limits_{i\le k<j}\{t[i][k]+t[k+1][j]+w(v_{i-1}v_kv_j)\} & i<j\end{cases}$$

代码实现

```
void MinweightTriangulation(int n,Type **t,int**s){
for(int i=1;i<= n; i++)
        t[i][i]= 0;
for(int r=2; r<=n; r++){
        for(int i=1; i<= n-r+1; i++){
                int j= i+r-1;
                t[i][j]= t[i+1][j]+ w(i-1, i , j);
                s[i][j]= i;
                for(int k=i+1; k<i+r-1; k++){
                        int u=t[i][k]+t[k+1][j]+w(i-1, k, j);
                        if(u<t[i][j]){
                                t[i][j]= u;
                                s[i][j]= k;
}}}}}
```

仿矩阵连乘积问题
复杂性：时间$O(n^3)$，空间$O(n^2)$