第9章　分枝限界方法

本章首先介绍分枝限界方法的基本思想及子集树和排列树的搜索等，并构建子集类问题和排列类问题分枝限界算法的公式化描述。然后介绍旅行商问题、最大团问题、0/1背包问题等问题的分枝限界方法算法，最后对优先级的确定进行分析和总结。

- 分枝限界方法的基本思想
- 排列树和子集树的搜索
- 旅行商问题的分枝限界算法
- 最大团问题的分枝限界算法
- 0/1背包问题的分枝限界算法*
- 启发式搜索

9.1 分枝限界方法的基本思想

9.1.1 分枝限界方法与回溯方法的比较

分枝限界方法同回溯方法类似，也是在解空间中搜索问题的可行解或最优解，但搜索方式不同。

回溯方法采用深度优先的方式，朝纵深方向搜索，直至达到问题的一个可行解，或经判断沿此路径不会达到问题的可行解或最优解时，停止向前搜索，并返回到该路径上最后一个可扩展结点。从该结点出发朝新的方向纵深搜索。

分枝限界方法采用宽度优先的方式搜索解空间树，将活结点存放在一个特殊的表中。在扩展结点处，先生成所有儿子结点,将那些导致不可行解或非最优解的儿子舍弃，其余儿子加入活结点表。此后，从活结点表中按照某种规则取出一个结点作为当前扩展结点，继续搜索。

9.1.2 分类

从活结点表中选择下一扩展结点的不同方式导致不同的分枝限界方法。最常见的有队列式分枝限界方法和优先队列式分枝限界方法两种方式。

1. 队列式分枝限界方法

将活结点表组织成一个队列，按先进先出原则选取下一个结点作为当前扩展结点。队列式分枝限界方法的搜索方式类似于解空间树的宽度优先搜索，但是队列式分枝限界方法不搜索不可行结点(不可能导致可行解或最优解的结点)为根的子树，这样的结点不放入活结点表。

2. 优先队列式分枝限界方法

将活结点表组织成一个优先队列，给结点规定优先级，选取优先级最高的下一个结点作为当前扩展结点。优先队列式分枝限界方法的搜索方式根据活结点的优先级确定下一个扩展结点。结点的优先级通常用一个与该结点有关的数值 p 来表示。最大优先队列规定 p 值较大的结点优先级较高，通常使用一个最大堆来实现。最小优先队列规定 p 值较小的结点优先级较高，通常使用一个最小堆来实现。

9.2 子集树和排列树的搜索

为了构造子集树搜索类问题和排列树搜索类问题的公式化描述，本节引入了虚的合法性检查函数legal(t)和界限检查函数bound(t)，值为true。

9.2.1 子集树

1. 虚的剪枝函数

将活结点表组织成一个队列Q，队列元素包含两个分量，X表示解向量，记录每个元素是否选取，t是待处理元素。

```
#include "algorithm.h"
bool legal(int t) { return true; } // 检查合法性
bool bound(int t) { return true; } // 检查界限
```

2. 队列元素类型

将活结点表组织成一个队列Q，队列元素包含两个分量，X表示解向量，记录每个元素是否选取，t是待处理元素。

```
struct SetNode // 队列元素类型
{   vector<bool> X; // 解向量
    int t; // 待处理元素
};
```

3. 算法描述

```
void BB_SubSet(int n) // 求子集的队列式分支限界法
{   queue<SetNode> Q; // 队列
    vector<bool>X(n); // 解向量，记录每个元素是否选取
    int t = 0; // 从元素0开始
    Q.push({X, t}); // 当前结点插入队列
    while(!Q.empty()) // 搜索子集树
    {   auto[X, t] = Q.front(); Q.pop(); // 取出扩展结点
        if(t >= n) cout << X << endl; // 到达叶子，输出答案
        else // 未到达叶子，产生孩子
        {   if(legal(t)) // 左儿子检查合法性后插入队列
                X[t] = 1, Q.push({X, t + 1});
            if(bound(t)) // 右儿子检查界限后插入队列
                X[t] = 0, Q.push({X, t + 1});
        }
    }
}
```

开始时创建一个队列。第1个扩展结点是子集树中的根结点。在该结点处，还没有确定子集的任何元素。因此，初始时有 $t=0$。算法的循环体完成对子集树内部结点的扩展，循环的终止条件是队列为空。对于当前扩展结点，算法分下列2种情况进行处理。

① 当 $t \geq n$ 时，当前扩展结点是子集树的一个叶子。

② 当 $t < n$ 时，依次产生当前扩展结点的所有儿子。当前扩展结点对应的子集是 $x[0..(t-1)]$，其左右儿子分别是 $x[t]=1$ 和 $x[t]=0$。若左儿子可行，则将左儿子插入活结点队列；若右儿子满足限界函数，则将右儿子插入活结点队列。

4. 测试程序*

```
int main()
{   BB_SubSet(3);
}
```

1 1 1 1 1 0	1 0 1 1 0 0	0 1 1 0 1 0	0 0 1 0 0 0

9.2.2 排列树

1. 虚的剪枝函数

将活结点表组织成一个队列Q，队列元素包含两个分量，X表示解向量，记录每个元素是否选取，t是待处理元素。

```
#include "algorithm.h"
bool legal(int t) { return true; } // 检查合法性
bool bound(int t) { return true; } // 检查界限
```

2. 队列元素类型

将活结点表组织成一个队列Q，队列元素包含两个分量，X表示解向量，记录排列中每个元素的值，t是待处理元素。

```
struct PermNode // 队列元素类型
{   vector<int> X; // 解向量
    int t; // 待处理元素
};
```

3. 算法描述

```
void BB_Perm(int n) // 求排列的队列式分支限界法
{   queue<PermNode> Q; // 队列
    vector<int> X(n); // 解向量，记录每个元素的值
    iota(begin(X), end(X), 0); // 解向量初始为自然排列
    int t = 0; // 从元素0开始处理
    Q.push({X, t}); // 将当前结点插入队列
    while(!Q.empty()) // 搜索排列树
    {   auto[X, t] = Q.front(); Q.pop(); // 取出扩展结点
        if(t >= n) cout << X << endl; // 到达叶子，输出答案
        else // 未到达叶子，产生孩子
            for(int i = t; i < n; ++i)
                if(legal(t) && bound(t)) // 检查可行性和界限
                {   swap(X[t], X[i]); // 交换X的第t号和第i号元素
                    Q.push({X, t + 1}); // 新结点插入队列
                }
    }
}
```

开始时创建一个队列。第1个扩展结点是排列树中的根结点。在该结点处，还没有确定排列的任何元素。因此，初始时有 $t=0$, $x[0..(n-1)]=[0..(n-1)]$ 。算法的循环体完成对排列树内部结点的扩展，循环的终止条件是队列为空。对于当前扩展结点，算法分下列2种情况进行处理。

① 当 $t \geq n$ 时，当前扩展结点是排列树的一个叶子。

② 当 $t<n$ 时，依次产生当前扩展结点的所有儿子。由于当前扩展结点对应的部分排列是 $x[0..(t-1)]$，其可行儿子只能是从剩余元素 $x[t..(n-1)]$ 中选取的元素 $x[i]$。将每一个满足限界函数的可行儿子插入活结点队列。

4. 测试程序*

```
int main()
{   BB_Perm(3);
}
```

0 1 2 0 2 1	1 0 2 1 2 0	2 1 0 2 0 1

9.3 旅行商问题

某无向加权图如图9-1左上部分所示，其解空间树是一棵排列树，如图9-1右下部分所示（图中结点左侧的数字表示该结点的费用，叶结点下侧的数字表示相应回路的费用，连线上的数字表示加权图中的顶点）。

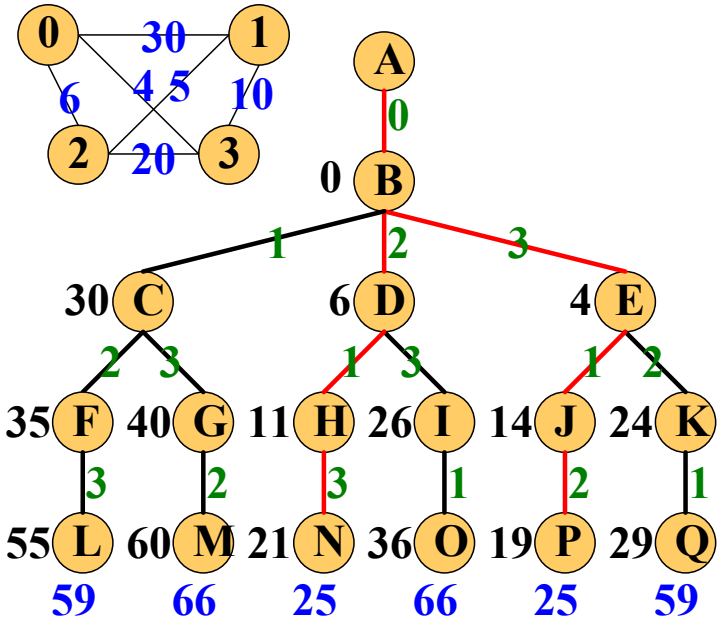
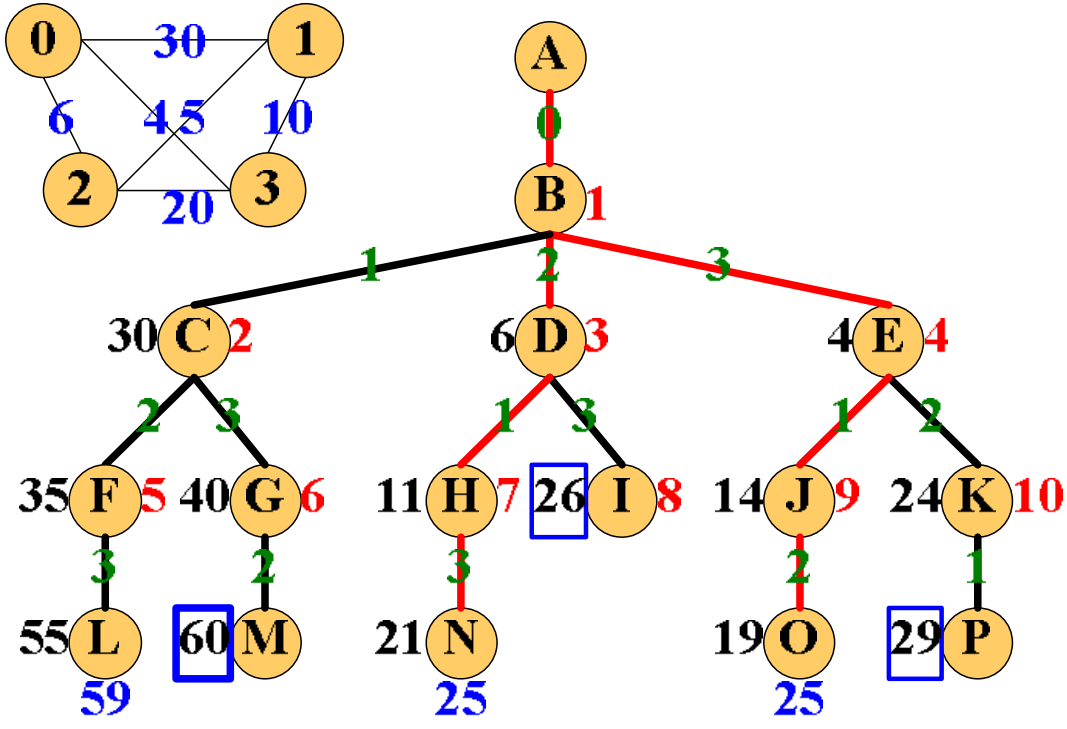


图9-1 TSP解空间

9.3.1　队列式分枝限界方法

上述TSP实例的队列式分枝限界算法的执行过程如图9-2所示。



{C, D, E}
{D, E, F, G}
{E, F, G, H, I}
{F, G, H, I, J, K}
{G, H, I, J, K, L}
{H, I, J, K, L, M}
{I, J, K, L, M, N}
{J, K, L, M, N}
{K, L, M, N, O}
{L, M, N, O, P}
{M, N, O, P}
{N, O, P}
{O, P}
{P}

图9-2　TSP的队列式分枝限界算法

以排列树中的结点 B 作为初始扩展结点。由于从顶点0到顶点1、2和3均有边相连，B 的儿子 C,D,E 都是可行结点，依次加入活结点队列，并舍弃当前扩展结点 B 。当前活结点队列中的队首结点 C 成为下一个扩展结点。由于顶点1到顶点2和3有边相连，故结点 C 的二个儿子 F,G 均为可行结点，加入活结点队列。接下来，结点 D,E 相继成为扩展结点。此时活结点队列中的结点依次为 F,G,H,I,J,K 。结点 F 成为下一个扩展结点，但其儿子是解空间树的叶结点，找到了一条TSP回路(0, 1, 2, 3, 0)，其费用为59。下一扩展结点 G 的儿子也是叶结点，得到另一条TSP回路(0, 1, 3, 2, 0)，费用为66。结点 H 成为当前扩展结点，其儿子也是叶结点，得到第三条TSP回路，费用为25，这是当前最短的TSP回路。下一个扩展结点是 I ，由于从根结点到结点 I 的费用26已经超过当前最优值，故没有必要扩展 I ，以 I 为根的子树被剪掉。最后依次扩展 J 和 K ，活结点队列为空，算法终止。算法搜索到的最优值是25，相应最优解是路径(0, 2, 1, 3, 0)。

9.3.2　优先队列式分枝限界方法

1. 执行过程

上述例子的优先队列式分枝限界算法的执行过程如图9-3所示。

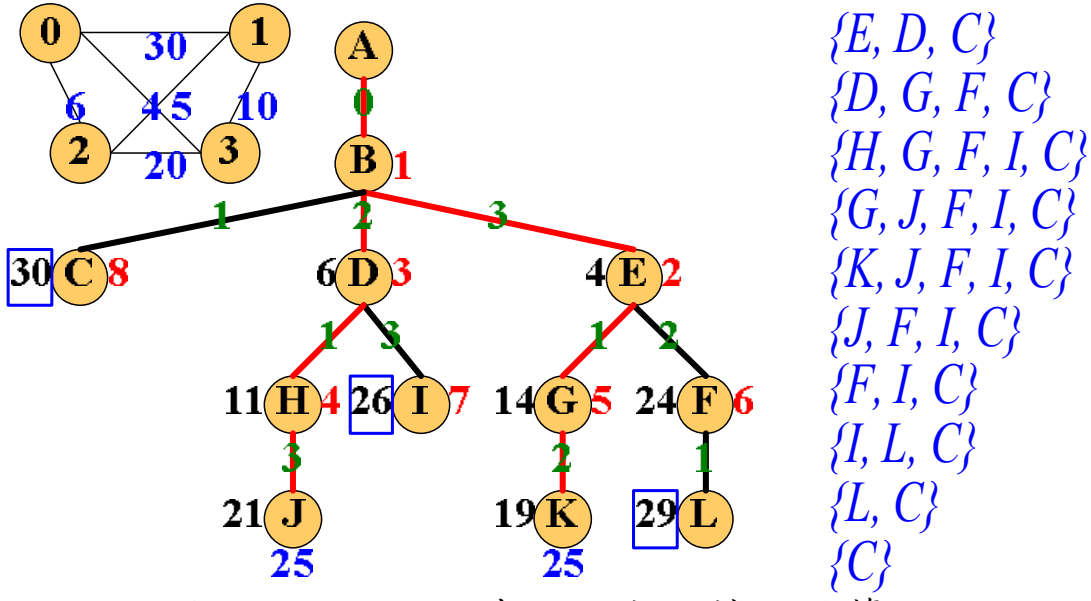


图9-3　TSP的优先队列式分枝限界算法

用一个最小堆存储活结点表，优先级函数是结点的当前费用。以排列树的结点 B 和空优先队列开始。结点 B 扩展后，它的3个儿子 C,D,E 依次插入堆中。此时，E 是堆中当前费用最小（为4）的结点，成为下一个扩展结点。结点 E 扩展后，其儿子 G,F 插入堆中，费用分别为14和24。此时，堆顶元素是 D，成为下一个扩展结点。它的2个儿子 H,I 插入堆中。此时，堆中元素是结点 H,G,F,I,C 。在这些结点中，H 费用最小，成为下一个扩展结点。扩展结点 H 后得到一条TSP回路(0, 2, 1, 3, 0)，相应费用为25。接下来，结点 G 成为扩展结点，并由此得到一条TSP回路(0, 3, 1, 2, 0)，费用仍为25。此后的两个扩展结点是 F 和 I 。由结点 F 得到的TSP回路的费用高于当前所知最小费用，结点 I 当前的费用已经高于当前所知最小费用，因而，它们都不能得到最优解。最后，优先队列为空，算法终止。

2. 队列元素类型

将活结点表组织成一个优先队列（最小堆）H，队列元素包含3个分量，X表示解向量，记录路径中的每个顶点，初始值为自然排列，t是待处理元素（下一站），C是当前费用。

```
#include "algorithm.h"
```

```
struct TSPNode // 队列元素类型
{   vector<int> X; // 解向量
    int t; // 待处理站
    double C; // 当前费用
};
```

```
bool operator<(const TSPNode &X, const TSPNode &Y)
{   return X.C < Y.C; // 用于最小堆, 比较当前费用
}
```

3. 算法描述

```
auto BB_TSP(const Matrix<double> &G)
{   int n = G.Rows(); // 顶点个数
    vector<int> X(n), BX; // 到当前结点的路径, 最优路径
    iota(begin(X), end(X), 0); // X初始为自然排列
    double C = 0, BC = inf; // 当前耗费和最优耗费, 初始为0和无穷大
    int t = 1; //出发站不参与排列, 下一站为1站
    minheap<TSPNode> H; // 优先队列(最小堆)
    H.push({X, t, C}); // 当前结点入队
```

```
    while(!H.empty()) // 搜索排列树
    {   auto [X, t, C] = H.top(); H.pop(); // 取出扩展结点
        double LC = C + G[X[n - 1]][0]; // 新的回路
        if(t >= n && LC < BC) // 当前旅行更优
            BC = LC, BX = X; // 更新最优旅行
        else if(t < n) // 未到达叶子，产生孩子
            for(int i = t; i < n; ++i)
            {   double CC = C + G[X[t - 1]][X[i]]; // 新的当前费用
                if(CC < BC) // 可能有更优旅行
                {   swap(X[t], X[i]); // 在新结点中交换站t和站i
                    H.push({X, t + 1, CC}); // 可行结点入队
                }
            }
    }
    return BX;
}
```

开始时创建一个最小堆，用于表示活结点优先队列。堆中每个结点的当前费用就是优先队列的优先级。

第1个扩展结点是排列树中根结点的唯一儿子。在该结点处，已确定的回路中只有顶点0。因此，初始时有 $t=0$，$x[0]=0$，$x[1..(n-1)]=[1..(n-1)]$，$c=0$。用 bc 记录当前最优值，初始时 $\mathrm{bc}=\inf$。

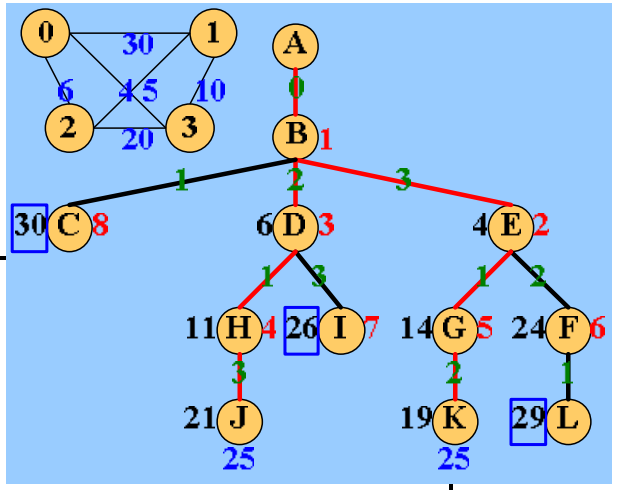
算法的循环体完成对排列树内部结点的扩展，循环的终止条件是队列为空，结束时返回找到的最小费用，相应最优解由数组 bx 给出。对于当前扩展结点，算法分下列2种情况进行处理。

① 当 $t\geq n$ 时，当前扩展结点是排列树中的叶子。如果该结点对应一条可行回路，且费用小于当前最优费用，则修改当前最优费用和当前最优回路。

② 当 $t<n$ 时，依次产生当前扩展结点的所有儿子。由于当前扩展结点对应的路径是 $x[0..(t-1)]$，其可行儿子是从剩余顶点 $x[t..(n-1)]$ 中选取的顶点 $x[i]$，且 $(x[t-1],x[i])$ 是一条边。对于每一个可行儿子，计算 $(x[0..(t-1)],x[i])$ 的费用 c。当 $c<\mathrm{bc}$ 时，将这个可行儿子插入活结点优先队列。

4. 测试程序*

使用如图9-1左上部分所示的图。




```
int main()
{   Matrix<double> G =
    {   {0, 30, 6, 4},
        {30, 0, 5, 10},
        {6, 5, 0, 20},
        {4, 10, 20, 0}
    };
    cout << "最优路径: " << BB_TSP(G) << endl;
}
```

```
最优路径: 0 3 1 2
```

{E, D, C}
{D, G, F, C}
{H, G, F, I, C}
{G, J, F, I, C}
{K, J, F, I, C}
{J, F, I, C}
{F, I, C}
{I, L, C}
{L, C}
{C}

9.4 最大团问题

从一个无向图中选取一个最大团。显然，最大团问题的解空间树是子集树。这里使用优先队列式分枝限界算法求解该问题。

9.4.1 剪枝函数

1. 可行性函数

可行性函数规定为顶点 t 到每一个已选顶点都有边相连，可由下列Connected()函数实现（保存在文件Clique.h中）。

```
bool Connected(const Matrix<bool> &G, const auto &X, int t)
{   for(int u = 0; u < t; ++u) // 检查顶点t与当前团的连接性, legal
        if(X[u] == 1 && G[t][u] == 0) return false;
    return true;
}
```

2. 上界函数

上界函数规定为有足够多的可选择顶点使得算法有可能找到更大的团。

用变量 cn 表示与该结点相应的团的顶点数；t 表示结点在解空间树中所处的层次，即当前正在处理的顶点（编号从0开始）；用 cn$+n-t$ 作为顶点数上界 un 的值（实际上可通过规定初始 un 为 n，生成右儿子时减1得到）。

在此优先队列式分枝限界算法中，un 实际上也是优先队列（最大堆）中元素的优先级。算法总是从活结点优先队列中抽取具有最大 un 值的元素作为下一个扩展元素。

9.4.2 分枝限界算法

1. 算法思想

解空间树的根结点是初始扩展结点，对于这个特殊的扩展结点，其 cn 值为0。算法在扩展内部结点时，首先考察其左儿子。在左儿子处，将顶点 t 加入当前团，并检查该顶点与当前团中其他顶点之间是否有边相连。若顶点 t 与当前团中所有顶点之间都有边相连，则相应的左儿子是可行结点，将它加入解空间树并插入活结点队列，否则就不是可行结点。接着继续考察当前扩展结点的右儿子。当 $\mathrm{un} > \mathrm{fn}$ 时，右子树中可能含有最优解，此时将右儿子加入解空间树并插入活结点队列。

2. 队列元素类型

将活结点表组织成一个优先队列（最大堆）H，队列元素包含4个分量，X表示解向量（当前团），记录每个顶点是否选取，t是待处理顶点，cn是当前团的大小，un团大小的上界。

```
#include "Clique.h"
#include "algorithm.h"
```

```
struct CliqueNode
{   vector<bool> X; // 解向量
    int t, cn, un; // 待处理顶点, 当前团大小, 团大小上界
};
```

```
bool operator<(const CliqueNode &X, const CliqueNode &Y)
{   return X.un < Y.un; // 用于最大堆, 比较团大小上界
}
```

3. 算法描述

```
auto BB_Clique(Matrix<bool> &G)
{   int n = G.Rows();
    vector<bool> X(n), BX; // 当前团, 最大团
    int cn = 0, fn = 0, un = n; // 当前大小, 最优大小, 大小上界
    int t = 0; // 从顶点0开始处理
    maxheap<CliqueNode> H; // 优先队列(最大堆)
    H.push({X, t, cn, un}); // 当前结点插入队列
    while(!H.empty())
    {   auto [X, t, cn, un] = H.top(); H.pop(); // 取出扩展结点
        if(t >= n && cn > fn) fn = cn, BX = X; // 到达叶子且有更优解
        else if(t < n) // 未到达叶子, 产生孩子
        {   if(Connected(G, X, t)) // t与当前团相连, 生成左儿子
                X[t] = 1, H.push({X, t + 1, cn + 1, un});
            if(un > fn) // 右子树可能更优
                X[t] = 0, H.push({X, t + 1, cn, un - 1});
        }
    }
    return BX;
}
```

4. 测试程序*

使用如图9-4所示的图。

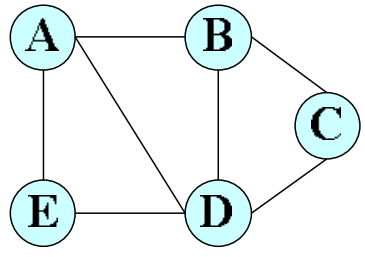


图9-4　一个有5个顶点的图

```
int main()
{   Matrix<bool> G = // 图的邻接矩阵, 图有5个顶点
    {   {0, 1, 0, 1, 1},
        {1, 0, 1, 1, 0},
        {0, 1, 0, 1, 0},
        {1, 1, 1, 0, 1},
        {1, 0, 0, 1, 0}
    };
    cout << "最大团: " << BB_Clique(G) << endl;
}
```

```
最大团: 1 1 0 1 0
```

9.5 0/1背包问题*

9.5.1 问题描述

有 n 件物品，第 i 件的重量和价值分别是 w_i 和 v_i。要将这 n 件物品的一部分装入容量为 c 的背包中，要求每件物品或整个装入或不装入。0/1背包问题就是要给出装包算法，使得装入背包的物品的总价值最大。本节采用分枝限界方法求解，选择出满足目标函数极大化和限定条件的物品的一个子集。

9.5.2 限界函数

在回溯方法一章中已经介绍，这里只回顾一下上界的获得方法，细节不再重述。如果在结点 Z 处已确定了 x_i 的值，$0 \le i \le k$，则 Z 的上界可由下述方法获得。

① 对 $k+1 \le i \le n-1$，将 $x_i \in \{0,1\}$ 的要求放宽为 $x_i \in [0,1]$。

② 用贪心方法求解这个放宽要求的问题（保存在文件KNAP.h中）。

③ 当前已获得的效益与贪心方法所得结果的和作为 Z 的上界。

9.5.3 算法设计

1. 队列元素类型

将活结点表组织成一个优先队列（最大堆）H，队列元素包含5个分量，X表示解向量，记录每个物品是否选取，t是待处理物品，cp是当前效益，cw是当前重量，up是效益的上界函数值。

```
#include "KNAP.h"
#include "algorithm.h"
```

```
struct KnapNode // 队列元素类型
{   vector<bool> X; // 解向量
    int t; // 待处理物品
    double cp, cw, up; // 当前效益，当前重量，上限函数值
};
```

```
bool operator<(const KnapNode &X, const KnapNode &Y)
{   return X.up < Y.up; // 用于最大堆，比较上限值
}
```

2. 算法描述

由上述界限函数可以得出，解空间树中每个结点的可行左儿子的上界与该结点的上界相同。因此，算法向左儿子移动时，不需要调用限界函数。

```
auto BB_BKNAP(double V[], double W[], int n, double M)
{   // 效益数组, 重量数组, 物品总数, 背包容量
    auto Rest = [&](int t, double rc) { return RestVal(V, W, n, rc, t); };
    // RestVal(V, W, n, rc, t)简写为Rest(t, rc)
    vector<bool> X(n), BX; // 解向量, 最优解
    double cp = 0, cw = 0, fp = -1, up = Rest(0, M);
    // 当前效益, 当前重量, 最大效益, 上限函数值
    Sort(V, W, n); // 物品数组按单价降序排列
    int t = 0; // 从物品0开始处理
    maxheap<KnapNode> H; // 最大堆H
    H.push({X, t, cp, cw, up}); // 当前结点插入队列
```

```
function RestVal(V, W, n, rc, t)
{   rv = 0;
    for(; t < n and W[t] <= rc; ++t)
        rv += V[t], rc -= W[t];
    if(t < n)
        rv += rc * V[t] / W[t];
    return rv;
}
```

```
    while(!H.empty()) // 搜索子集树
    {   auto [X, t, cp, cw, up] = H.top(); H.pop(); // 取出扩展结点
        if(t >= n && cp > fp) // 到达叶子且有更优解
            fp = cp, BX = X; // 更新最优值和最优解
        else if(t < n) // 未到达叶子，产生孩子
        {   if(cw + W[t] <= M) // 左儿子可行，物品放入背包
            {   X[t] = 1;
                H.push({X, t + 1, cp + V[t], cw + W[t], up});
            }
            if(up >= fp) // 根t子树上限大于fp，生成右儿子
            {   X[t] = 0;
                H.push({X, t + 1, cp, cw, cp + Rest(t + 1, M - cw)});
            }
        }
    }
    return BX;
}
```

3. 测试程序*

```
int main()
{   int n = 3; // 物品数量
    double V[n] = {120, 60, 100}; // 效益数组
    double W[n] = {30, 10, 20}; // 重量数组
    double M = 50; // 背包容量
    auto X= BB_BKNAP(V, W, n, M);
    PrintSolution(V, W, X, n);
}
```

```
效益: 60 100 120
重量: 10 20 30
答案: 0 1 1
```

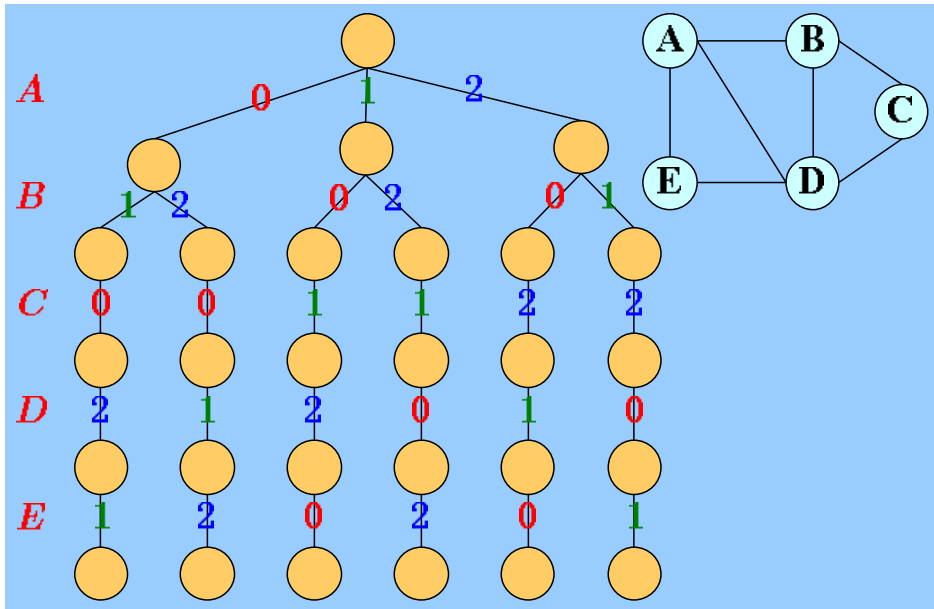
9.6 图的着色问题*

图的着色问题是给定无向图 G 和 m 种颜色，求出 G 的所有 m-着色。

9.6.1 解空间与剪枝函数

1. 解空间

用邻接矩阵 W 表示图 G，$W[i,j]=1$ 表示顶点 i 与 j 相邻（有边相连），否则 $W[i,j]=0$。m 种颜色分别用 $0,1,\ldots,m-1$ 表示。易知，解空间是一个完全 $m-$ 叉树，共 $n+1$ 级。



2. 剪枝函数

图 G 的一种 m - 着色用一个数组 X 表示，$X[i]=k$ 表示顶点 i 被着上颜色 k。X 是可行解，当且仅当

$$\text{若 } W[i,j]=1\text{，则 } X[i]\neq X[j]$$

这就是约束条件。

假如前 t 个顶点已经着色，即 $X[0],\ldots,X[t-1]$ 已经确定，则确定顶点 t 的颜色 $X[t]$ 时，根据约束条件，应该验证

$$\text{若 } W[i,t]=1\text{，则 } X[i]\neq X[t],\ i=0,\ldots,t-1$$

是否满足。

9.6.2 分支限界算法

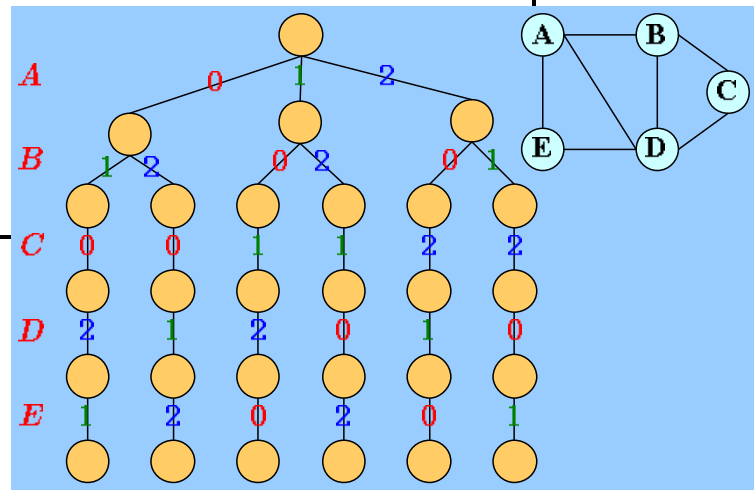
1. 队列元素类型

```
#include "GraphColor.h"
#include "algorithm.h"
```

```
struct ColorNode
{   vector<int> X; // 解向量
    int t; // 待处理顶点
};
```

2. 算法描述

```
void BB_Color(Matrix<bool> &G, int m) // 邻接矩阵, 颜色(1~m)
{   int n = G.Rows(); // 顶点数
    vector<int> X(n); // 当前着色方案
    int t = 0; // 从顶点0开始处理
    queue<ColorNode> Q; // 队列
    Q.push({X, t}); // 当前结点插入队列
    while(!Q.empty())
    {   auto[X, t] = Q.front(); Q.pop(); // 取出扩展结点
        if(t >= n) cout << X << endl; // 到达叶子
        else if(t < n) // 未到达叶子, 产生孩子
        {   for(X[t] = 0; X[t] < m; ++X[t])
            {   if(legal(G, X, t)) // 颜色X[t]可用, 考虑下一顶点
                    Q.push({X, t + 1});
            }
        }
    }
}
```



3. 举例说明

$n=5, m=3$，无向图 G 如图8-6所示。

```
int main()
{   int m = 3; // 颜色数
    Matrix<bool> G = // 邻接矩阵
    {   {0, 1, 0, 1, 1},
        {1, 0, 1, 1, 0},
        {0, 1, 0, 1, 0},
        {1, 1, 1, 0, 1},
        {1, 0, 0, 1, 0}
    };
    BB_Color(G, m);
}
```

```
0 1 0 2 1
0 2 0 1 2
1 0 1 2 0
1 2 1 0 2
2 0 2 1 0
2 1 2 0 1
```

9.7 启发式搜索

优先队列式分枝限界也可以称为启发式搜索。在这种算法中，结点优先级的确定直接影响算法性能。如果优先级函数规定较小数值为较高优先级，则对应的搜索策略称为最小成本检索，简称为 LC - 检索。如果优先级函数规定较大数值为较高优先级，则对应的搜索策略称为最大效益检索。这里以最小成本检索为例讨论优先级的确定。

9.7.1 优先级的确定

对于最小成本检索，理想的当前扩展结点 X 应该满足下列条件。

① 以 X 为根的子树中含有问题的答案。

② 在所有满足①的活结点中， X 距离答案“最近”。

用 $c(X)$ 表示一个理想的优先级函数，定义如下。

① 如果 X 是答案，则 $c(X)$ 是解空间树中，由根到 X 的成本（即所用的代价，如级数、计算复杂度等）。

② 如果 X 不是答案，且以 X 为根的子树中不含答案，则 $c(X)$ 定义为 ∞ 。

③ 如果 X 不是答案，但是以 X 为根的子树中含有答案，则 $c(X)$ 是成本最小的答案的成本。

9.7.2 最小成本检索

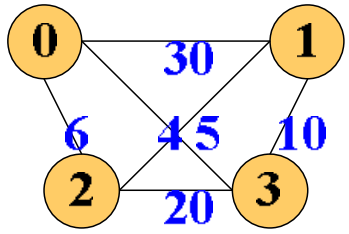
如果能够获得理想的优先级函数，则优先队列式分枝限界算法将以最少的时间找到问题的解。然而，这样的优先级函数的计算工作量不亚于解原问题。实际工作中采用的是成本估计函数 $\hat{c}(X)$，由两部分决定：解空间树中根到 X 的成本 $f(X)$ 和 X 到答案的估计成本 $g(X)$，即 $\hat{c}(X)=f(X)+g(X)$。选取 $\hat{c}(X)$ 值最小的活结点作为下一个扩展结点。这种搜索策略称为最小成本检索，简称为 LC - 检索。在前述旅行商问题中，规定 $f(X)=cc$， $g(X)=0$。

9.7.3 最大效益检索

最小成本检索对于“优先级函数规定较小数值为较高优先级”的情况非常适用。如果优先级函数规定较大数值为较高优先级，则可以对称地讨论所谓“最大效益检索”。在前述最大团问题中，规定 $f(X)=\mathrm{cn}$， $g(X)=n-t$。在前述0/1背包问题中，规定 $f(X)=\mathrm{cp}$， $g(X)=\mathrm{RestVal}(t+1)$。

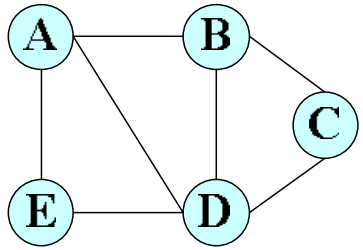
9.8 练习题

1、分别给出采用队列式分枝限界方法和优先队列式分枝限界方法求解如图所示的旅行商问题（从0出发）的解空间树和活结点队列的变化过程。



2、使用一种程序设计语言或伪代码描述旅行商问题和最大团问题的分枝限界算法。

3、给出采用优先队列式分枝限界方法求解如图所示的最大团问题解空间树。



4、使用一种程序设计语言或伪代码描述求解下列子集和问题的分支限界算法，并分析其时间复杂性。

设 S 是 n 个正整数构成的集合，t 是一个正整数，寻找 S 的满足 $\sum_{x\in S'} x \le t$ 且使得 $\sum_{x\in S'} x$ 最大的子集 S'。

5、使用一种程序设计语言或伪代码描述用分枝限界方法求解下列问题的算法并分析时间复杂性。

皇后问题（输出所有解），Hamilton回路问题（输出所有Hamilton回路），旅行商问题（输出最优解），子集和问题（输出所有解），最大团问题（输出最优解），0/1背包问题（输出最优解）