湖南科技大学课程教案
（章节、专题首页）

授课教师：王志喜　　职称：副教授　　单位：计算机科学与工程学院

课 程 名 称	算法设计与分析
章节、专题	分枝限界方法
教学目标及基 本 要 求	熟悉分枝限界方法的基本思想，能够使用分枝限界方法求解各种子集类问题和排列类问题。
教 学 重 点	排列树和子集树的搜索，旅行商问题和最大团问题的分枝限界算法
教 学 难 点	旅行商问题和0/1背包问题的分枝限界算法
教学内容与时 间 分 配	（1）分枝限界方法的基本思想（0.4课时） （2）排列树和子集树的搜索（0.7课时） （3）旅行商问题的分枝限界算法（0.4课时） （4）最大团问题的分枝限界算法（0.3课时） （5）0/1背包问题的分枝限界算法（0.3课时） （6）着色问题的分枝限界算法（0.2课时） （7）启发式搜索（0.2课时） 共计2.5课时。
习　　题	第8.8.1节（基础知识题），第8.8.2节（实验题）。

第8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背包问题和着色问题等问题的分枝限界算法，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
- 着色问题的分枝限界算法
- 启发式搜索

8.1 分枝限界方法的基本思想

8.1.1 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可能达到问题的可行解或最优解时，停止向前搜索。然后，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。然后，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。

8.1.2 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，最常见的分枝限界方法是队列式分枝限界方法和优先队列式分枝限界方法。

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。结点的优先级通常用一个与该结点有关的数值 p 表示。最大优先队列规定 p 值较大的结点优先级较高，通常使用最大堆实现。最小优先队列规定 p 值较小的结点优先级较高，通常使用最小堆实现。

8.1.3 分枝限界方法的一般流程

1. 队列元素类型

将活结点表组织成一个队列或一个优先队列，队列元素通常包含两个分量，X表示解向量，记录答案中每个分量的值，t表示待处理元素。

2. 部分说明

这里的核心任务是在确定解向量X的前t个分量$X[0..(t-1)]$的值以后，如何确定分量$X[t]$的值。为了方便描述，引入下列两个函数。

- $T(X[0..(t-1)])$。表示在确定$X[0..(t-1)]$以后，$X[t]$所有可能取值的集合。
- $B(X[0..t])$。表示$X[0..t]$是否违背约束条件，其中，$B(X[0..t])$ = true表示$X[0..t]$没有违背约束条件，$B(X[0..t])$ = false表示$X[0..t]$违背了约束条件。

3. 流程描述

（1）变量声明及初始化。

```
function Branch()
{   Q = {}; // 准备一个空队列
    InitValue(X); // 解向量各分量的初始值必须根据具体问题指定
    t = 0; // 从0开始
    Q.push({X, t}); // 当前结点入队
```

（2）宽度优先遍历解空间树。

```
    while(not Q.empty()) // 队列非空
    {   [X, t] = Q.pop(); // 取出扩展结点
        if(IsAnswer(X)) Print X; // 答案
        else if(t < n)
            for(X[t] in T(X[0 .. (t - 1)]))
                if(B(X[0 .. t])) // 激活状态结点X[0 .. t], 生成儿子
                    Q.push({X, t + 1}); // 新结点入队
    }
}
```

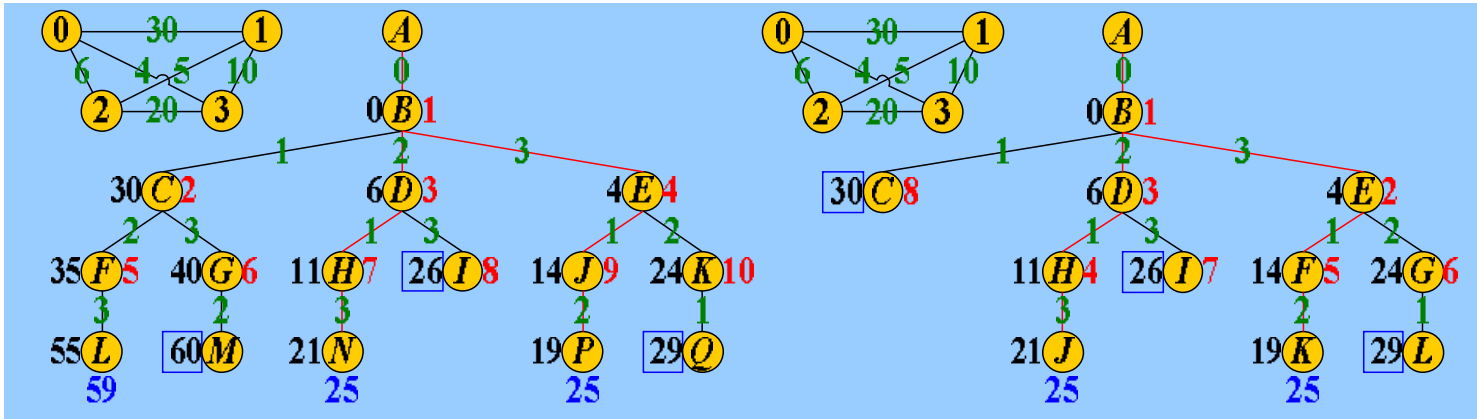
8.1.4 与回溯方法的比较

1. 复杂性估计

在分枝限界方法中，因为入队和出队等操作需要复制解向量等相关对象，而且在优先队列进行调整时还需要频繁移动解向量等相关对象，这些操作不能常数时间完成，所以分枝限界算法的复杂性估计可能会高于回溯算法。

2. 优先级

回溯算法不方便指定结点的优先级，而分枝限界算法能够很方便地指定结点的优先级。如果优先级选得比较理想，则分枝限界方法能够很快地找到一个比较好的答案，使得剪枝操作尽可能发生在解空间树的上层，能够最大可能地剪去不好的分枝。例如，对于图8-1所示的旅行商问题实例，分枝限界算法（右侧）能够比回溯算法（左侧）剪去更多的不好分枝。这里，优先级就选用当前扩展结点的成本。



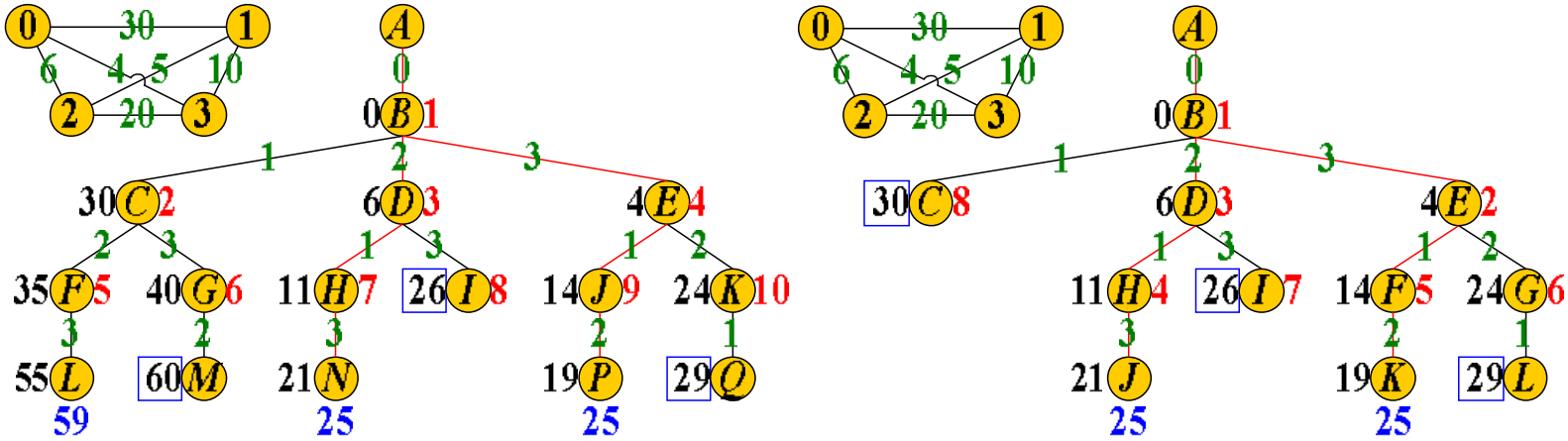


图8-1　分枝限界方法与回溯方法的比较

8.2 子集树和排列树的搜索

为了构造子集树（包括m-叉树）搜索类问题和排列树搜索类问题回溯算法的公式化描述，本节使用在“回溯方法”一章中引入的虚的可行性检查函数和界限检查函数。在算法描述中，也同样假定$X[0..n)$为解向量。

```
#include "algorithm.h"
bool legal(int t) { return true; }
bool bound(int t) { return true; }
```

8.2.1 m-叉树的搜索

1. 队列元素类型

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

```
#include "dummy.h"
struct MSetNode // 队列元素类型
{   vector<int> X; // 解向量
    int t; // 待处理元素
};
```

2. 算法描述

（1）变量声明及初始化。

```
void MSet(int n, int m) // 搜索m-叉树的队列式分枝限界算法
{   queue< MSetNode> Q; // 队列
    vector<int>X(n); // 解向量，记录每个元素的值
    int t = 0; // 从元素0开始
    Q.push({X, t}); // 当前结点入队
```

（2）宽度优先遍历解空间树。

```
    while(not Q.empty())
    {   auto[X, t] = Q.front(); Q.pop(); // 取出扩展结点
        if(t >= n) cout << X << endl; // 答案
        else
            for(X[t] = 1; X[t] <= m; ++X[t]) // X[t] in [1 .. m]
                if(legal(t) and bound(t)) // 可行和界限
                    Q.push({X, t + 1});
    }
}
```

8.2.2 子集树的搜索

1. 队列元素类型

将活结点表组织成一个队列，队列元素包含两个分量，*X*表示解向量，记录原集合中每个元素是否选入当前子集，*t*是待处理元素。

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

2. 算法描述

（1）变量声明及初始化。

```
void SubSet(int n) // 求子集的队列式分枝限界法
{   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}); // 当前结点入队
```

（2）宽度优先遍历解空间树。

```
    while(not 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; // 答案
        else
        {   X[t] = 1;
            if(legal(t)) Q.push({X, t + 1}); // 可行
            X[t] = 0;
            if(bound(t)) Q.push({X, t + 1}); // 界限
        }
    }
}
```

8.2.3 排列树的搜索

1. 队列元素类型

将活结点表组织成一个队列，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。

```
#include "dummy.h" // 虚的剪枝函数
struct PermNode // 队列元素类型
{   vector<int> X; // 解向量
    int t; // 待处理元素
};
```

2. 算法描述

（1）变量声明及初始化。

```
void Perm(int n) // 求排列的队列式分枝限界法
{   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); // 解向量需初始化为自然排列
    int t = 0; // 从元素0开始处理
    Q.push({X, t}); // 当前结点入队
```

（2）宽度优先遍历解空间树。

```
    while(not 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; // 答案
        else
            for(int i = t; i < n; ++i)
            {   swap(X[t], X[i]);
                if(legal(t) and bound(t)) // 可行和界限
                    Q.push({X, t + 1});
            }
    }
}
```

8.3 旅行商问题

8.3.1 答案、剪枝和优先级

1. 答案

若当前扩展结点是解空间树的叶结点，则找到了一条TSP回路。若该回路的费用比当前最优值小，则将答案更改为该回路并修改当前最优值。

2. 剪枝函数

（1）可行性函数。若当前边的费用已经超过当前最优值，则没有必要扩展该结点，以该结点为根的子树被剪掉。

（2）界限函数。若从根结点到当前扩展结点的费用（当前费用）已经超过当前最优值，则没有必要扩展该结点，以该结点为根的子树被剪掉。

显然，满足界限条件可以保证满足可行性条件，没有必要专门检查可行性。

3. 优先级

使用优先队列式分枝限界算法，选用当前费用C作为队列元素的优先级。算法总是从活结点队列中抽取C值最小的结点作为下一扩展结点。

8.3.2 分枝限界算法

1. 队列元素类型

将活结点表组织成一个优先队列（最小堆），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; // 当前旅行
    int t; // 待处理站
    double C; // 当前费用
};
```

选用当前费用作为队列元素的优先级。

```
bool operator<(const TSPNode &X, const TSPNode &Y)
{   return X.C < Y.C; // 优先级为当前费用
}
```

2. 算法描述

（1）变量声明及初始化。

```
auto TSP(const Matrix<double> &G) // 邻接矩阵
{   minheap<TSPNode> H; // 优先队列(最小堆)
    int n = G.rows(); // 顶点数
    vector<int> X(n), BX; // 到当前结点的路径, 最优路径
    iota(begin(X), end(X), 0); // 解向量需初始化为自然排列
    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
    H.push({X, t, C}); // 当前结点入队
```

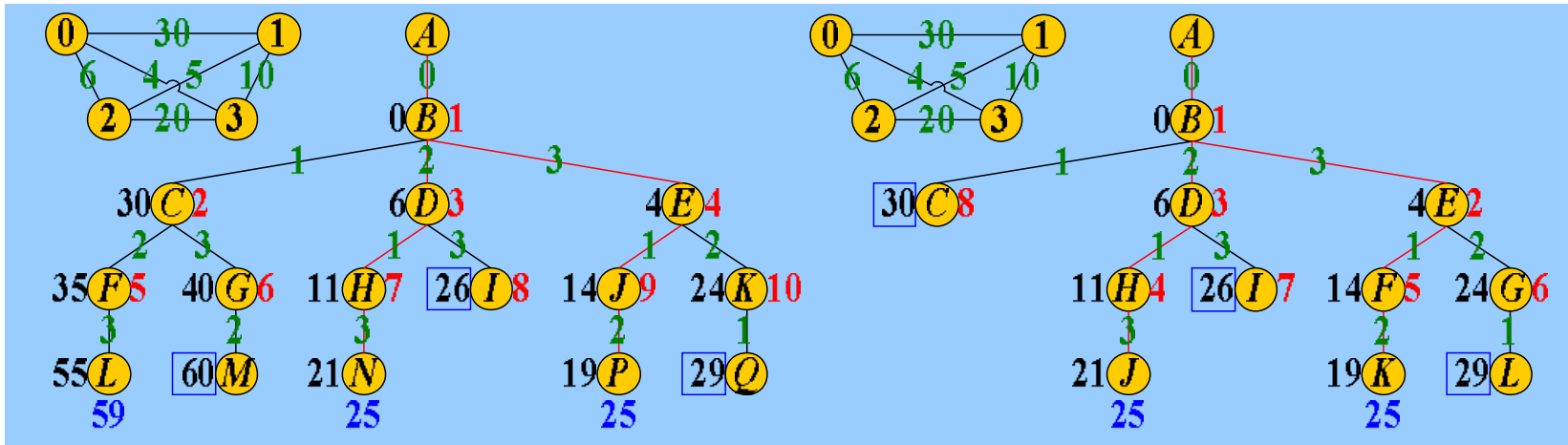
（2）宽度优先遍历解空间树。

```
    while(not H.empty()) // 搜索排列树
    {   auto [X, t, C] = H.top(); H.pop(); // 取出扩展结点
        if(t >= n and C + G(X[n - 1], 0) < BC) // 答案(当前回路更优)
            BC = C + G(X[n - 1], 0), BX = X;
        else if(t < n)
            for(int i = t; i < n; ++i)
            {   swap(X[t], X[i]);
                // 检查可行和界限
                if(C + G(X[t - 1], X[t]) < BC) // 当前路径可能更优
                    H.push({X, t + 1, C + G(X[t - 1], X[t])}); // 新结点入队
            }
    }
    return BX;
}
```

3. 测试程序

使用如图8-1所示的图。

```
int main()
{   Matrix<double> G = //  邻接矩阵
    {   {0, 30, 6, 4},
        {30, 0, 5, 10},
        {6, 5, 0, 20},
        {4, 10, 20, 0}
    };
    cout << TSP(G) << endl; // 0 3 1 2
}
```



```
bool Connected(const Matrix<bool> &G,
     const vector<bool> &X, int t)
{    int n = G.rows();
     for(int u = 0; u < n; ++u)
     {    if(u == t) continue;
          if(X[u] == 1 and G(t, u) == 0)
               return false;
     }
     return true;
}
```

8.4 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。

8.4.1 剪枝和优先级

1. 可行性函数

检查顶点t与当前团是否相连，具体实现请参阅“回溯方法”一章。

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。

如果用变量cn表示结点对应的团的大小，t表示结点在解空间树中的层次，即当前正在处理的顶点（编号从0开始），则可用cn+n-(t+1)作为顶点数上界值。

3. 优先级

使用优先队列式分枝限界算法，选用当前团大小cn作为队列元素的优先级。

8.4.2 分枝限界算法

1. 队列元素类型

将活结点表组织成一个优先队列（最大堆），队列元素包含3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。

```
#include "Clique.h"
struct CliqueNode
{   vector<bool> X; // 当前团(需初始化为空集)
    int t, cn; // 待处理顶点, 当前团大小
};
```

选用当前团大小作为队列元素的优先级。

```
bool operator<(const CliqueNode &X, const CliqueNode &Y)
{   return X.cn < Y.cn; // 优先级为当前团大小
}
```

2. 算法描述

（1）变量声明及初始化。

```
auto Clique(const Matrix<bool> &G)
{   maxheap<CliqueNode> H; // 最大堆
    int n = G.rows(); // 顶点数
    vector<bool> X(n, 0),   BX; // 当前团(初始为空集), 最大团
    int cn = 0, fn = 0; // 当前团顶点数, 最优团顶点数
    int t = 0; // 从顶点0开始处理
    H.push({X, t, cn}); // 当前结点入队
```

（2）宽度优先遍历解空间树。

```
    while(not H.empty())
    {   auto [X, t, cn] = H.top(); H.pop(); // 取出扩展结点
        if(t >= n and cn > fn) // 答案(更优)
            fn = cn, BX = X;
        else if(t < n)
        {   X[t] = 1;
            if(Connected(G, X, t)) // 可行(当前成团)
                H.push({X, t + 1, cn + 1}); // 新结点入队
            X[t] = 0;
            if(cn + n - (t + 1) > fn) // 界限(可能有更大的团)
                H.push({X, t + 1, cn}); // 新结点入队
        }
    }
    return BX;
}
```

3. 测试程序

使用如图8-2所示的图。

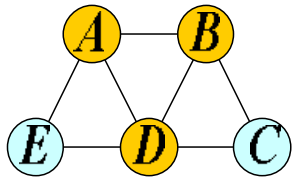


图8-2　最大团

```
int main()
{   Matrix<bool> G = // 邻接矩阵(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 << Clique(G) << endl; // 1 1 0 1 0
}
```

```
function RestVal(V, W, rc, t)
{   // n是物品数
    rv = 0; // 剩余部分最优效益
    for(; t < n and W[t] <= rc; ++t)
        rv += V[t], rc -= W[t]; // 可完整装包
    if(t < n) rv += rc * V[t] / W[t];
    return rv;
}
```

8.5 0/1背包问题

8.5.1 问题分析

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。从中选取一部分物品装入容量为c的背包中，每件物品只能整体装入或不装入。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。

2. 限界函数

如果在结点Z处已经确定了x_i的值（$0 \le i \le t$），则 Z 的上界可由下述方法获得（具体实现请参阅“回溯方法”一章）。

（1）对$t+1 \le i < n$，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。

（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。

3. 优先级

使用优先队列式分枝限界算法，选用当前效益作为队列元素的优先级。

8.5.2 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），队列元素包含4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。

```
#include "Knap.h"
struct KnapNode // 队列元素类型
{   vector<bool> X; // 当前装包方案
    int t; // 待处理物品
    double cv, cw; // 当前效益, 当前重量
};
```

选用当前效益上限作为队列元素的优先级。

```
bool operator<(const KnapNode &X, const KnapNode &Y)
{   return X.cv < Y.cv; // 优先级为当前效益
}
```

2. 算法描述

（1）变量声明及初始化。

```
auto Knap(vector<double> &V, vector<double> &W, double M)
{   // 效益数组, 重量数组, 背包容量
    Sort(V, W); // 物品数组按单价降序排列
    maxheap<KnapNode> H; // 最大堆H
    int n = min(V.size(), W.size()); // 物品数
    vector<bool> X(n), BX; // 当前装包方案, 最优装包方案
    double cv = 0, cw = 0, fv = -1; // 当前效益, 当前重量, 最优效益
    int t = 0; // 从物品0开始处理
    H.push({X, t, cv, cw}); // 当前结点入队
```

（2）宽度优先遍历解空间树。

```
    while(not H.empty()) // 搜索子集树
    {   auto [X, t, cv, cw] = H.top(); H.pop(); // 取出扩展结点
        if(t >= n and cv > fv) // 答案(更优)
            fv = cv, BX = X;
        else if(t < n)
        {   X[t] = 1;
            if(cw + W[t] <= M) // 可行(当前不超重), 新结点入队
                H.push({X, t + 1, cv + V[t], cw + W[t]});
            X[t] = 0;
            auto up = cv + RestVal(V, W, M - cw, t + 1); // 上限值
            if(up > fv) // 界限(可能有更优的装包), 新结点入队
                H.push({X, t + 1, cv, cw});
        }
    }
    return BX;
}
```

```
function RestVal(V, W, rc, t)
{   // n是物品数
    rv = 0; // 剩余部分最优效益
    for(; t < n and W[t] <= rc; ++t)
        rv += V[t], rc -= W[t]; // 可完整装包
    if(t < n) rv += rc * V[t] / W[t];
    return rv;
}
```

3. 测试程序

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

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

```
function legal(G,X, t)
{   for(i = 0; i < t; ++i)
        if(G(i, t) == 1 and X[i] == X[t])
            return false;
    return true;
}
```

8.6 着色问题

本节讨论最优化形式的着色问题。

8.6.1 问题分析

1. 问题描述

为一个需要着色的无向图选取一种颜色数目最少的着色方案。

2. 可行性函数

若前 t 个顶点的颜色 $X_0,\ldots,X_{t-1}$ 已经确定，则在确定顶点 t 的颜色 X_t 时，应该验证是否有“对任何 $i\in[0\ ..\ t)$，若 $G(t,i)=1$，则 $X_i\neq X_t$”。具体实现请参阅“回溯方法”一章。

3. 界限函数

当前方案部分顶点已用颜色的数量比已知较优着色方案的颜色数少，从而有可能获得颜色数更少的着色方案。

4. 优先级

使用优先队列式分枝限界算法，选用当前方案部分顶点已用颜色的数量与待考虑的剩余顶点数之和（颜色数上限，即假定剩余顶点均需使用一种新颜色）作为队列元素的优先级。

8.6.2 分枝限界算法

1. 队列元素类型

将活结点表组织成一个优先队列（最小堆），队列元素包含4个分量，*X*表示解向量，记录每个顶点的颜色，*t*是待处理顶点，*m*是当前方案部分顶点已用颜色的数量，up是颜色数上界（已用顶点数与剩余顶点数之和）。

```
#include "Chromatic.h"
struct ChromaticNode
{   vector<int> X; // 当前着色方案, 即各顶点颜色(1, 2, ..., m)
    int t, m, up; // 当前顶点, 已用颜色数(界限), 颜色数上限(优先级)
};
```

选用已用颜色数与剩余顶点数之和（颜色数上限）作为队列元素的优先级。

```
bool operator<(const ChromaticNode &X, const ChromaticNode &Y)
{   return X.up < Y.up; // 优先级为颜色数上限
}
```

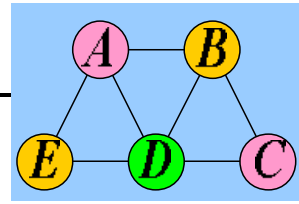
2. 算法描述

（1）变量声明及初始化。

```
auto Chromatic(Matrix<bool> &G) // 邻接矩阵
{   minheap<ChromaticNode> H; // 最小堆
    int n = G.rows(); // 顶点数
    vector<int> X(n, -1), BX; // 当前方案(初始时未着色), 最优方案
    int fm = n + 1; // 最优颜色数
    H.push({X, 0, 0, n}); // 从顶点0开始, 初始颜色数为0, 上限为n
```

（2）宽度优先遍历解空间树。

```
    while(not empty(H)) // 搜索解空间树
    {   auto [X, t, m, up] = H.top(); H.pop(); // X, 当前顶点, 颜色数, 上限
        if(t >= n and m < fm) // 答案(颜色数更小)
            BX = X, fm = m;
        else if(t < n)
            for(X[t] = 1; X[t] <= m + 1; ++X[t]) // 有新顶点, 需增加颜色
            {   auto cm = max(X[t], m); // 新的颜色数
                auto up = cm + n - (t + 1); // 优先级(颜色数上限)
                if(legal(G, X, t) and cm < fm) // X[t]可用且颜色数可能更小
                    H.push({X, t + 1, cm, up}); // 下一顶点
            }
    }
    return BX;
}
```



3. 测试程序☼

$n=5, m=3$，无向图 G 如图8-2所示。

```
int main()
{   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}
    };
    cout << Chromatic(G) << endl; // 1 2 1 3 2
}
```

8.7 启发式搜索※

优先队列式分枝限界搜索是一种启发式搜索。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。

8.7.1 优先级的确定

1. 理想的扩展结点

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”。

2. 理想的优先级函数

用$c(X)$表示一个理想的优先级函数，定义如下。

- 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）。
- 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∞。
- 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。

8.7.2 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 $g(X)$ 这两部分决定，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。

在本章给出的分枝限界算法示例中，对于旅行商问题，规定 $f(X)=\mathrm{cc}$，$g(X)=0$，对于着色问题，规定 $f(X)=\mathrm{cm}$，$g(X)=n-(t+1)$。

8.7.3 最大效益检索

最小成本检索适用于“优先级函数规定较小数值为较高优先级”的情况。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”。

在本章给出的分枝限界算法示例中，对于最大团问题，规定 $f(X)=\mathrm{cn}$，$g(X)=0$，对于0/1背包问题，规定 $f(X)=\mathrm{cv}$，$g(X)=0$。

8.7.4 界限函数和优先级的选用

通常，按照下列原则选用界限函数和优先级函数，平均情况下的剪枝效果通常不会太差。

（1）对于最小成本检索，界限函数和优先级函数都可以选用当前成本。如果能够比较方便地估计成本上限，则优先级函数通常选用成本上限。例如，在本章给出的分枝限界算法示例中，对于旅行商问题，界限函数和优先级都规定为当前费用，对于着色问题，界限函数规定为已用颜色数，优先级规定为颜色数上限。

（2）对于最大效益检索，通常能够比较方便地估计效益上限。此时，界限函数通常选用效益上限，优先级通常选用当前效益。例如，在本章给出的分枝限界算法示例中，对于最大团问题，界限函数规定为团顶点数上限，优先级规定为当前团顶点数，对于0/1背包问题，界限函数规定为效益上限，优先级规定为当前效益。

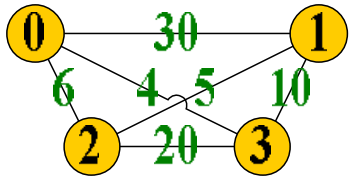
8.8 练习题

8.8.1 基础训练题

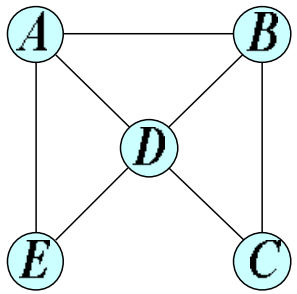
1. 编写一个使用分枝限界方法生成含n个分量的所有排列的程序。

2. 编写一个使用分枝限界方法生成n个元素的所有子集的程序。

3. 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。



4. 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。



5. 使用一种程序设计语言描述求解旅行商问题(输出最优解)的分枝限界算法，并分析时间复杂性。

6. 使用一种程序设计语言描述求解最大团问题(输出最优解)的分枝限界算法，并分析时间复杂性。

7. 使用一种程序设计语言描述求解子集和问题(输出所有解)的分枝限界算法，并分析时间复杂性。

8.8.2 实验题

1. 使用一种程序设计语言描述求解Hamilton回路问题（输出所有Hamilton回路）的分枝限界算法，并分析时间复杂性。

2. 使用分枝限界方法求解Hamilton回路问题，要求只输出一条Hamilton回路，或报告不存在Hamilton回路。

3. 使用一种程序设计语言描述求解0/1背包问题（输出最优解）的分枝限界算法，并分析时间复杂性。

4. 使用分枝限界方法求解子集和问题，要求只输出一个满足条件的子集，或报告不存在满足条件的子集。

5. 请使用分枝限界方法为一个需要着色的无向图选取一种颜色数目最少的着色方案。

8.8.3 进阶训练题

1. 设S是一个正整数集合，t是一个正整数，寻找S的满足子集和不大于t且使得子集和最小的子集。为该问题设计分枝限界算法。

2. 设S是一个正整数集合，t是一个正整数，寻找S的满足子集和不小于t且使得子集和最小的子集。为该问题设计分枝限界算法。

3. 顶点覆盖问题的分枝限界算法。为最小顶点覆盖问题（输出一个最小顶点覆盖）设计分枝限界算法。

4. 集合覆盖问题的分枝限界算法。已知一个有限集X和X的一组子集F，如果F中所有子集的并等于X，则称F覆盖X。请使用分枝限界方法从F中选取一组覆盖X且子集数目最少的子集。

5. 精确覆盖问题的分枝限界算法。有限集X的集合覆盖F中覆盖X的互不相交的子集族T称为$<X, F>$的精确覆盖。为精确覆盖问题（$<X, F>$是否有精确覆盖）设计分枝限界算法。

6. 子集和问题的分枝限界算法。使用分枝限界方法从一个正整数集W中选取一个和最接近正整数t的子集。

7. 均衡划分问题的分枝限界算法。为均衡划分问题（将一个正整数集分为和最

接近的两个子集）设计分枝限界算法。

8. 多机调度问题的分枝限界算法。为多机调度问题（输出一种最优调度）设计分枝限界算法。

9. CNF-SAT的分枝限界算法。为CNF-SAT设计分枝限界算法，用于计算给定合取范式的最大可满足子句数，并输出一种最优赋值。

10. 考查课选题问题的分枝限界算法。某一门考查课的任课老师准备了n道考核候选题，每位考生选取m道题完成，要求每两个考生的选题方案之间至少有k道题是不相同的，请问这些候选题能够保证多少考生能够顺利按照要求完成选题。为该问题设计分枝限界算法。

11. 考查课出题问题的分枝限界算法。某一门考查课的考核方法是由任课老师准备一些考核候选题，每位考生选取m道题完成，要求每两个考生的选题方案之间至少有k道题是不相同的，请问任课老师至少需要准备多少道候选题才能保证n个考生能够顺利按照要求完成选题。为该问题设计分枝限界算法。