回溯整体的模板其实只有一套：

```
void dfs(输入参数)
{
        if(终止条件)
        {
                返回答案;
                return;
        }

        for(开始进行选择，子树大小就是选择目标集合大小)
        {
                if(能够被选中的条件)
                {
                        进行操作，选上之后更新参数
                        dfs();//进行下一层递归
                        复原现场
                }
        }

}
```

回溯法与分治限界显著的不同就是回溯法中的所有节点都可能不止一次成为活节点，而分治限界必然只有一次机会成为活节点。
同样的回溯法中的活节点可以不止一次成为扩展节点，而分治限界法只能一次。

解空间树

解空间树通常有两类。一类是子集树，即从大集合中选择一部分作为最终答案；还有一种是排列树，即解空间中的元素全是集合元素不同的排列顺序，所有元素都要被用到。

活节点，扩展节点，死节点

- 我们目前到达额节点，并进行操作的节点成为扩展节点。
- 当前扩展了节点，并且节点符合条件，但是还未进行任何操作的节点称为活节点
- 已经被确认不符合约束条件的，并且不再需要进行扩展的节点称为死节点。

装载问题/0 1背包

老问题，对于每一个货物，我们都分别选择装与不装。如果不装的话总重小于最优解则剪枝。

```
# include<bits/stdc++.h>
using namespace std;
typedef long long ll;

vector<int>ans;
vector<int>found;
int n,v;
int t[114514];//体积
int w[114514];
int best;
int have;//价值
int load;
int maxen;

bool st[115414];

void dfs(int now)
{

        if(have+maxen<=best)
        return;

        if(now==n)
        {
                if(have>best)
                {
                        best=have;
                        ans=found;
                }
                return;
        }

        for(int i=0;i<n;i++)
        {
                if(st[i]==false&&load+t[i]<=v)
                {
                        found.push_back(i);
                        have+=w[i];
                        load+=t[i];
                        st[i]=true;
                        maxen-=w[i];
                        dfs(now+1);
                        st[i]=false;
```

```
                        found.pop_back();
                        have-=w[i];
                        load-=t[i];
                        maxen+=w[i];

                }
        }
        dfs(now+1);

}


int main (void)
{
        cin>>n>>v;
        for(int i=0;i<n;i++)
        {
        cin>>t[i]>>w[i];
        maxen+=w[i];

        }
        dfs(0);
        cout<<best<<endl;


        return  0;

}
```

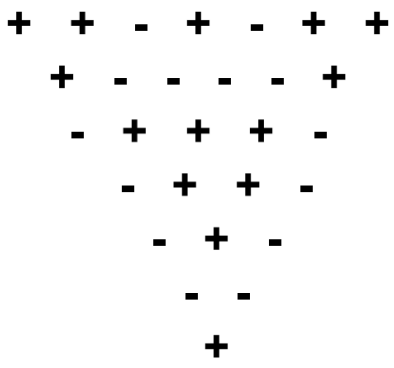
回溯法解背包和装载问题平均复杂度为$O(2^n)$

作业调度

首先生成所有作业的n种全排列。然后按顺序依次执行各项任务。处理时间计算方法：
对于作业x_i，看前一个作业x_{i-1}在机器2上完成的时间，取$max(x_{i-1}$在机器二上被完成时间，
x_i在机器一上需要时间加x_{i-1}在机器一上被完成时间)作为起点，加上x_i在任务二上的需要时间，作为该作业需

符号三角形

下图是由14个”+”和14个”-”组成的符号三角形。2个同号下面都是”+”，2个异号下面都是”-”。



在一般情况下，符号三角形的第一行有n个符号。符号三角形问题要求对于给定的n，计算有多少个不同的符号三角形，使其所含的”+”和”-”的个数相同。

如果第一行有n个符号，那么最后出来的三角形一共有$(n+1)*n/2$个符号。易得如果$(n+1)*n/2$是奇数的话可以直接不用计算，必然没有答案。如果是偶数就继续搜索。

每一个符号三角形都由第一行唯一确定。一旦第一行确定了那么后续就确定了。我们只需要列出第一行的各种可能情况然后按照第一行展开计算即可。

```
# include<bits/stdc++.h>
using namespace std;
typedef long long ll;

vector<int>first;
int arr[100][1000];
int n;
int cnt1;
int cnt2;
int ans;
bool judge(vector<int>have)
{
        cnt1=0;
        cnt2=0;
        for(int i=0;i<have.size();i++)
        {
                if(have[i]==1)
                cnt1++;
                else
```

```
                cnt2++;
        }
        for(int i=0;i<have.size();i++)
        arr[0][i]=have[i];

        int tempo=n;
        int now=0;
        while(tempo>=1)
        {
                for(int i=0;i<tempo-1;i++)
                {
                        if(arr[now][i]==arr[now][i+1])
                        {
                        arr[now+1][i]=1;
                        cnt1++;
                        }
                        else
                        {
                        arr[now+1][i]=0;cnt2++;
                        }
                }
                now++;
                tempo--;
                if(cnt1>(n+1)*n/4||cnt2>(n+1)*n/4)
                return false;
        }
        if(cnt1==cnt2)
        return true;
        else
        return false;


}
void  dfs(int t)
{

        if(t==n)
        {
                if(judge(first))
                {
                        ans++;
                }
                return;
```

```
        }

        first.push_back(1);
        dfs(t+1);
        first.pop_back();
        first.push_back(0);
        dfs(t+1);
        first.pop_back();

}
int main (void)
{
        cin>>n;

        if((((n+1)*n/2)%2)!=0)
        {

                cout<<"无答案"<<endl;
                return 0;

        }
        dfs(0);
        cout<<ans<<endl;

        return 0;

}
```

复杂度为$O(n2^n)$

四色猜想

可以只用四个颜色就令一个图中所有相连的点不是同一个颜色。

思想为从一个点出发赋上一个颜色，然后给其他点赋上不同的颜色。每一次赋色判断是否合法即可。实现简单。

假设一共有m种颜色可供选择，有n个点，那么时间复杂度为$O(nm^n)$

旅行售货员

见优化版dijkstra。