回溯法

5-5 设G是一个有n个顶点的有向图。

从顶点i发出的边的最大费用记为 max(i)。

(1) 证明旅行售货员回路的费用不超过 $\sum_{i=1}^{n} max(i)+1$。

任一旅行售货员回路 $(\pi_1, \pi_2, \cdots, \pi_n)$ 的费用是

$a(\pi_1, \pi_2) + a(\pi_2, \pi_3) + \cdots + a(\pi_{n-1}, \pi_n) + a(\pi_n, \pi_1)$

显然，$a(\pi_k, \pi_{k+1}) \leq max(\pi_k)$，故其费用 $\leq \sum_{i=1}^{n} max(\pi_i) < \sum_{i=1}^{n} max(i)+1$

(2) 在回溯法中用上面的界作为 bestc 的初始值，重写该算法并尽可能简化代码。

```
void Traveling<T>::Backtrack(int i){
  if(i==n){
    if(a[x[n-1]][x[n]] != NoEdge && a[x[n]][1] != NoEdge &
     (cc+a[x[n-1]][x[n]] + a[x[n]][1]< bestc)){
      for(int j=1;j<=n; j++){
        bestx[j]=x[j]; }
      bestc = cc+ a[x[n-1]][x[n]]+ a[x[n]][1];
  }
  }
  else{
    for(int j=i; j<=n; j++){
      if(a[x[i-1]][x[j]] != NoEdge && (cc+ a[x[i-1]]
      [x[j]]< bestc)){
        Swap(x[i], x[j]);
```

```
            cc+=a[x[i-1]][x[i]];
            Backtrack(i+1);
            cc-=a[x[i-1]][x[i]];
            Swap(x[i], x[j]);
        }
    }
}
```

即删去了原代码中 bestc == NoEdge 的判断，但 bestc 初始值定为 $\sum_{i=1}^{n} \max(i)+1$.

5-6 (1) 证明图G的所有前缀为 $x[1:i]$ 的旅行售货员回路的费用至少为：$\sum_{j=2}^{i} a(x_{j-1}, x_j) + \sum_{j=i}^{n} \min(x_j)$

前缀为 $x[1:i]$ 的任一回路 $(x_1, x_2, \cdots, x_i, \pi_{i+1}, \cdots, \pi_n)$ 的总费用为

$$\begin{aligned}
&\sum_{j=2}^{i} a(x_{j-1}, x_j) + a(x_i, \pi_{i+1}) + a(\pi_{i+1}, \pi_{i+2}) + \cdots \\
&+ a(\pi_{n-1}, \pi_n) + a(\pi_n, x_1) \\
&\geq \sum_{j=2}^{i} a(x_{j-1}, x_j) + \min(x_i) + \min(\pi_{i+1}) + \cdots + \min(\pi_n) \\
&= \sum_{j=2}^{i} a(x_{j-1}, x_j) + \sum_{j=1}^{n} \min(x_j) .
\end{aligned}$$

(2) 设计一个计算最小出边费用和的函数 rcost()，在回溯函数的遍历部分，改写为

```
...
else{
```

```
for(int j=i; j<=n; j++){
    if(a[x[i-1]][x[j]]!=NoEdge && (cc+rcost())<bestc)){
        Swap(x[i],x[j]);
        ...
```

这样可以剪掉更多非最短路径的枝，提高算法效率。

5-2| 子集树问题

```
void Load<T>: Backtrack (int t){
if(t>n) Output();
else{   for(int i=0; i<=1; i++){
          x[t]=i;
        if(Constraint(t) && Bound(t)){
            Change(t);
            Backtrack(t+1);
            Restore(t);
}}}
```

```
void Load<T>: Output(){
    ii=n;  bestw=cw;  return;  }
bool Load<T>: Constraint(int t){
return x[t]==0||x[t]==1 && cw+w[t]<=c . }
bool Load<T>: Bound (int t){
return x[t]==1 || x[t]==0 && cw+r-w[t]>bestw}
void Load<T>: Change (int t){
    if(x[t]==1) cw+=w[t];
```

```
    r-=w[t]; }
void Load<T>::Restore(int t){
    if(x[t]==1) cw=w[t];
    r+=w[t];
    if(ii=t) bestx[t]=x[t];   ii--;   }
```