3.7. Задача оптимального выбора

Наш последний пример алгоритма поиска с возвратом является логическим развитием предыдущих двух в рамках общей схемы. Сначала мы применили принцип возврата, чтобы находить *одно* решение задачи. Примером послужили задачи о путешествии шахматного коня и о восьми ферзях. Затем мы разобрались с поиском *всех* решений; примерами послужили задачи о восьми ферзях и о стабильных браках. Теперь мы хотим искать *оптимальное* решение.

Для этого нужно генерировать все возможные решения, но выбрать лишь то, которое оптимально в каком-то конкретном смысле. Предполагая, что оптимальность определена с помощью функции f(s), принимающей положительные значения, получаем нужный алгоритм из общей схемы Try заменой операции *печатать решение* инструкцией

```
IF f(solution) > f(optimum) THEN optimum := solution END
```

Переменная optimum запоминает лучшее решение из до сих пор найденных. Естественно, ее нужно правильно инициализировать; кроме того, обычно значение f(optimum) хранят еще в одной переменной, чтобы избежать повторных вычислений.

Вот частный пример общей проблемы нахождения оптимального решения в некоторой задаче. Рассмотрим важную и часто встречающуюся проблему выбора оптимального набора (подмножества) из заданного множества объектов при наличии некоторых ограничений. Наборы, являющиеся допустимыми решениями, собираются постепенно посредством исследования отдельных объектов исходного множества. Процедура Try описывает процесс исследования одного объекта, и она вызывается рекурсивно (чтобы исследовать очередной объект) до тех пор, пока не будут исследованы все объекты.

Замечаем, что рассмотрение каждого объекта (такие объекты назывались кандидатами в предыдущих примерах) имеет два возможных исхода, а именно: либо исследуемый объект включается в собираемый набор, либо исключается из него. Поэтому использовать циклы repeat или for здесь неудобно, и вместо них можно просто явно описать два случая. Предполагая, что объекты пронумерованы 0, 1, ... , n-1, это можно выразить следующим образом:

```
PROCEDURE Try (i: INTEGER);
BEGIN
    IF i < n THEN
        IF включение допустимо THEN
            включить i-й объект;
            Try(i+1);
            исключить i-й объект
        END;
        IF исключение допустимо THEN
            Try(i+1)
        END
    ELSE
        проверить оптимальность
    END
END Try
```

Уже из этой схемы очевидно, что есть 2^n возможных подмножеств; ясно, что нужны подходящие критерии отбора, чтобы радикально уменьшить число исследуемых кандидатов. Чтобы прояснить этот процесс, возьмем конкретный пример задачи выбора: пусть каждый из n объектов $a_0, \dots, a_{n-1}$ характеризуется своим весом и ценностью. Пусть оптимальным считается тот набор, у которого суммарная ценность компонент является наибольшей, а ограничением пусть будет некоторый предел на их суммарный вес. Эта задача хорошо известна всем путешественникам, которые пакуют чемоданы, делая выбор из n предметов таким образом, чтобы их суммарная ценность была наибольшей, а суммарный вес не превышал некоторого предела.

Теперь можно принять решения о представлении описанных сведений в глобальных переменных. На основе приведенных соображений сделать выбор легко:

```
TYPE Object = RECORD weight, value: INTEGER END;
VAR a: ARRAY n OF Object;
    limw, totv, maxv: INTEGER;
    s, opts: SET
```

Переменные limw и totv обозначают предел для веса и суммарную ценность всех n объектов. Эти два значения постоянны на протяжении всего процесса выбора. Переменная s представляет текущее состояние собираемого набора объектов, в котором каждый объект представлен своим именем (индексом). Переменная opts – оптимальный набор среди исследованных к данному моменту, а maxv – его ценность.

Каковы критерии допустимости включения объекта в собираемый набор? Если речь о том, имеет ли смысл *включать* объект в набор, то критерий здесь – не будет ли при таком включении превышен лимит по весу. Если будет, то можно не добавлять новые объекты к текущему набору. Однако если речь об *исключении*, то допустимость дальнейшего исследования наборов, не содержащих этого элемента, определяется тем, может ли ценность таких наборов превысить значение для оптимума, найденного к данному моменту. И если не может, то продолжение поиска, хотя и может дать еще какое-нибудь решение, не приведет к улучшению уже найденного оптимума. Поэтому дальнейший поиск на этом пути бесполезен. Из этих двух условий можно определить величины, которые нужно вычислять на каждом шаге процесса выбора:

1. Полный вес tw набора s, собранного на данный момент.
2. Еще достижимая с набором s ценность av.

Эти два значения удобно представить параметрами процедуры Try. Теперь условие ***включение допустимо*** можно сформулирловать так:

```
tw + a[i].weight < limw
```

а последующую проверку оптимальности записать так:

```
IF av > maxv THEN (*новый оптимум, записать его*)
    opts := s; maxv := av
END
```

Последнее присваивание основано на том соображении, что когда все n объектов рассмотрены, достижимое значение совпадает с достигнутым. Условие ***исключение допустимо*** выражается так:

```
av - a[i].value > maxv
```

Для значения av - a[i].value, которое используется неоднократно, вводится имя av1, чтобы избежать его повторного вычисления.

Теперь вся процедура составляется из уже рассмотренных частей с добавлением подходящих операторов инициализации для глобальных переменных. Обратим внимание на легкость включения и исключения из множества s с помощью операций для типа SET. Результаты работы программы показаны в табл. 3.5.

```
TYPE Object = RECORD value, weight: INTEGER END; (* ADruS37_OptSelection *)
VAR a: ARRAY n OF Object;
    limw, totv, maxv: INTEGER;
    s, opts: SET;

PROCEDURE Try (i, tw, av: INTEGER);
    VAR tw1, av1: INTEGER;
BEGIN
    IF i < n THEN
        (*проверка включения*)
        tw1 := tw + a[i].weight;
        IF tw1 <= limw THEN
            s := s + {i};
            Try(i+1, tw1, av);
            s := s - {i}
        END;
        (*проверка исключения*)
        av1 := av - a[i].value;
        IF av1 > maxv THEN
            Try(i+1, tw, av1)
        END
    ELSIF av > maxv THEN
        maxv := av; opts := s
    END
END Try;
```

Таблица 3.5. Пример результатов работы программы Selection при выборе из 10 объектов (вверху). Звездочки отмечают объекты из отпимальных наборов **opts** для ограничений на суммарный вес от 10 до 120

вес:	10	11	12	13	14	15	16	17	18	19	
ценность:	18	20	17	19	25	21	27	23	25	24	
limw ↓											maxv
10	*										18
20							*				27
30					*		*				52
40	*				*		*				70
50	*	*		*			*				84
60	*	*	*	*	*						99
70	*	*			*		*		*		115
80	*	*	*		*		*	*			130
90	*	*			*		*		*	*	139
100	*	*		*	*		*	*	*		157
110	*	*	*	*	*	*	*		*		172
120	*	*			*	*	*	*	*	*	183

```
PROCEDURE Selection (WeightInc, WeightLimit: INTEGER);
BEGIN
    limw := 0;
    REPEAT
        limw := limw + WeightInc; maxv := 0;
        s := {}; opts := {}; Try(0, 0, totv);
    UNTIL limw >= WeightLimit
END Selection.
```

Такая схема поиска с возвратом, в которой используются ограничения для предотвращения избыточных блужданий по дереву поиска, называется *методом ветвей и границ* (branch and bound algorithm).