ема № 8. Жадные алгоритмы, теоретические основы, применение.

Дайте мне таблеток от жадности и побольше, побольше…

( Народный фольклор )

8.1 Введение.

В теории алгоритмов играют важную роль жадные алгоритмы. Они просты для понимания и реализации, работают сравнительно быстро, известно много разнообразных задач, которые можно решить с помощью таких алгоритмов. Однако не всегда можно доказать возможность применимости жадного алгоритма для нахождения точного решения многих задач.

**Жадный алгоритм**(greedy algorithm) - метод решения оптимизационных задач, основанный на том, что процесс принятия решения можно разбить на элементарные шаги, на каждом из которых принимается отдельное решение. Решение принимаемое на каждом шаге должно быть оптимальным только на текущем шаге и должно приниматься без учета предыдущих или последующих решений.

Алгоритмы, предназначенные для решения задач оптимизации, обычно представляют собой последовательность шагов, на каждом из которых предоставляется некоторое множество выборов. Определение наилучшего выбора, руководствуясь принципами динамического программирования, во многих задачах оптимизации напоминает стрельбу из пушки по воробьям; другими словами, для этих задач лучше подходят более простые и эффективные алгоритмы. В ***жадном алгоритме***всегда делается выбор, который кажется самым лучшим в данный момент - т.е. производится локально оптимальный выбор в надежде, что он приведет к оптимальному решению глобальной задачи. Жадные алгоритмы не всегда приводят к оптимальному решению, но во многих задачах они дают нужный результат. Жадныйалгоритмобладаетдостаточноймощью и хорошоподходитдля широкогоклассазадач. Алгоритмы поиска минимальных остовных деревьев являются классическим примером применения жадного метода.

Признаки того, что задачу возможно решить при помощи жадного алгоритма:

1.Задачу можно разбить на подзадачи;

2. Величины, рассматриваемые в задаче, можно дробить так же на подзадачи;

3.Сумма оптимальных решений для двух подзадач даст оптимальное решения для всей задачи.

**Пример 8.1** Пассажирский лифт не может поднять больше W кг. В лифт пытаются влезть H человек, причем для каждого из них известен его вес: W1, W2... WH. Определить какое максимальное количество людей смогут уехать на лифте за один раз.

**Решение .**Очевидно,чтоэлементарнойподзадачей является помещение в лифт одного человека. Если имеется несколько кандидатов на помещение в лифт, то оптимальным выбором будет человек с наименьшим весом, т.к. при этом остается наибольший запас по грузоподъемности. Поэтому, длярешениязадачи, отсортируем людей по их весу и будем, начиная с самого легкого, помещать их в лифт, пока это еще можно сделать.

**Пример 8.2 Задача о выборе заявок.**

Даны *n* заявокна проведение занятий в некоторой аудитории. В каждой заявке указаны начало и конец занятия (*si* и *fi* для *i*-й заявки). Два разных занятия не могут перекрываться по времени. Заявки с номерами *i* и *j* совместны, если интервалы [*si*, *fi*) и [*sj*, *fj*) не пересекаются (т. е. *fi* ≤ *sj* или *fj* ≤ *si*). Задача о выборе заявок состоит в том, чтобы набрать максимальное количество совместных друг с другом заявок.

Например, рассмотрим описанное ниже множество S процессов, отсортированных в порядке возрастания моментов окончания:

**Решение .** Приведем жадный алгоритм, решающий данную задачу. При этом полагаем, что заявки упорядочены в порядке возрастания времени окончания.

Activity-Selector(*s*,*f*)

1 *n* ← *length*[*s*]

2 *A* ← {1}

3 *j* ← 1

4 **for** *i* ← 2 **to** *n*

5 **do if** *si* ≥ *fj*

6 **then** *A* ← *A* ∪ {*i*}

7 *j* ← *i*

8 **return** *A*

Работа алгоритма показана на рис. 8.1

Процедура работает следующим образом. Переменная i индексирует самое последнее добавление к множеству A, соответствующее процессу ai в рекурсивной версии.

В строках 2–3 выбирается процесс a1, инициализируется множество A, содержащее только этот процесс, а переменной i присваивается индекс этого процесса. В цикле for в строках 4–7 происходит поиск процесса задачи Si,n+1, оканчивающегося раньше других. В этом цикле по очереди рассматривается каждый процесс aj который добавляется в множество A, если он совместим со всеми ранее выбранными процессами; этот процесс оканчивается раньше других в задаче Si,n+1. Чтобы узнать, совместим ли процесс ajс процессами, которые уже содержатся во множестве A, достаточно проверить (строка 5), что его начальный момент sjнаступает не раньше момента fi окончания последнего из добавленных в множество A процессов. Если процесс aj удовлетворяет сформулированным выше условиям, то в строках 6–7 он добавляется в множество A и переменной i присваивается значение j.

Алгоритм работает за O(*n*log2*n*+*n*), т. е. сортировка плюс выборка. На каждом шаге выбирается наилучшее решение. Покажем, что в итоге получится оптимум.

**Теорема 8.1**Алгоритм Activity-Selector(*s*,*f*) даёт набор из наибольшего возможного количества совместных заявок.

**Доказательство.** Заметим, что все заявки отсортированы по возрастанию времени окончания. Заявка номер 1, очевидно, входит в оптимум (если нет, то заменим самую раннюю заявку в оптимуме на нее, от этого хуже не станет). Выкинув все заявки, не совместные с первой, получим исходную задачу с меньшим количеством заявок. Рассуждая по индукции, аналогичным образом приходим к оптимальному решению.

