Методы построения алгоритмов

Литература

1. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: построение и анализ. — М.: Изд. Дом Вильямс, 2000. — 960 с.
2. Ахо А., Хопкрофт Д., Ульман Д. Структуры данных и алгоритмы. – М.: Изд. Дом Вильямс, 2000. – 384с.
3. Кнут Д. Искусство программирования, т.1 Основные алгоритмы, Изд. Дом Вильямс, 2000. – 384с.

Метод «разделяй и властвуй»

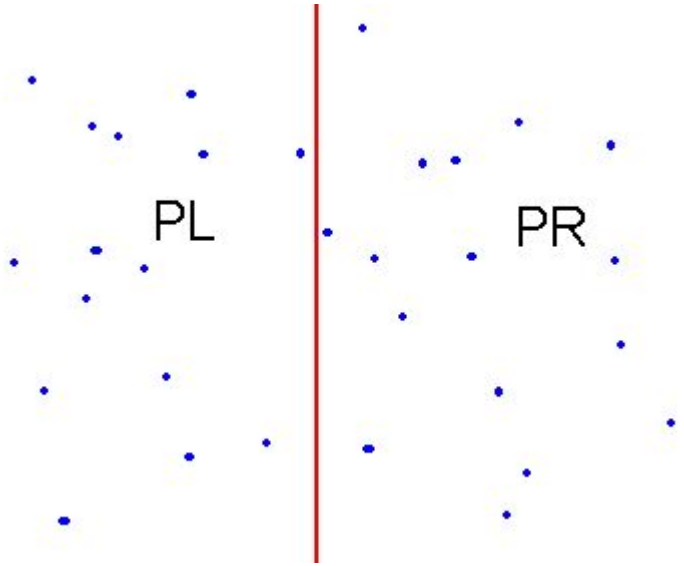
- *Метод декомпозиции* (или метод "разделяй и властвуй", или метод разбиения).
  - предполагает такую декомпозицию (разбиение) задачи размера *n* на более мелкие задачи, что на основе решений этих более мелких задач можно легко получить решение исходной задачи.
- Метод применяется:
  - в сортировке слиянием
  - в деревьях двоичного поиска

Метод «разделяй и властвуй»

Пример. Задача о ближайших точках

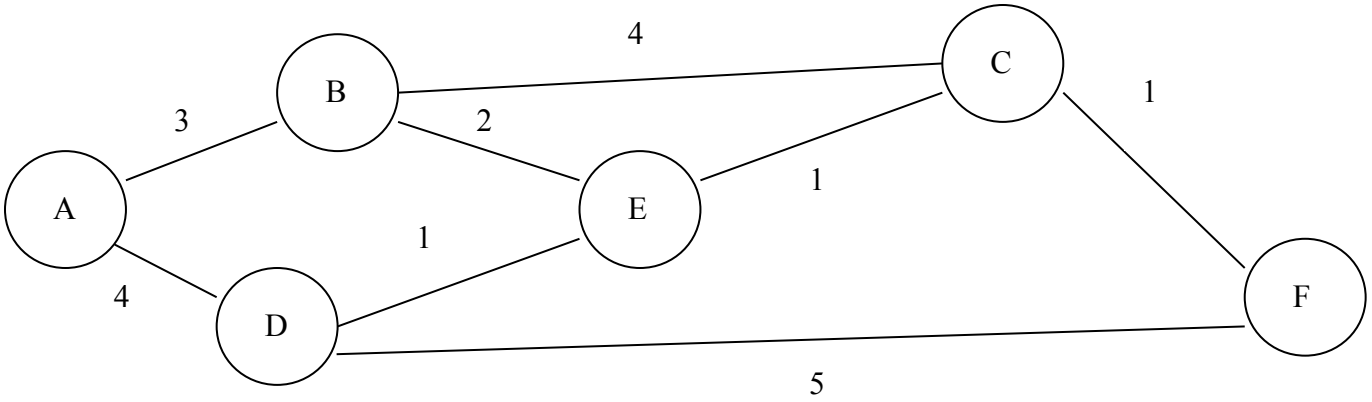
- **Постановка задачи**
  - Дано множество точек на плоскости. Найти среди них две ближайшие (с наименьшим расстоянием).
- **Метод решения задачи**
  - Алгоритм со временем работы $O(n\log n)$ может быть построен методом ***разделяй и властвуй***. Он имеет рекурсивную структуру
- **Входные данные для рекурсивной процедуры**
  - Входные данные для каждого рекурсивного вызова алгоритма состоят из некоторого множества p точек и двух массивов — X и Y.

- **Разделяй**



- **Властвуй**
  - **После деления выполняем два рекурсивных вызова — для левой и правой частей. Пусть *d*R и *d*L — результаты для левой и правой частей соответственно. *d* = min(*d*R, *d*L).**
- **Соединяй**
  - **Для всего множества *p* ответом будет либо *d* (и соответствующая ему пара), либо какая-то пара приграничных точек**

- ***Метод последовательных приближений***
  - Начиная с исходного приближения f0, производится последовательное уточнение решения f1, затем уточняется f2, f3 , f4 , f5 и т.д.
- ***Метод наискорейшего спуска***
  - принцип: «чтобы достигнуть дна, требуется только идти вниз».
    - **Пример.** Задача коммивояжера
      - МНС дает хорошее, но не обязательно оптимальное решение:
        - A – B – E – C – F – D – A



- ***Метод обратного прохода***
  - применяется тогда, когда задан порядок (направление) решения некоторой задачи. Замена этого направления на обратное может упростить задачу без ее изменения.
- **Пример**
  - Дано: два резервуара объемом 5 и 9 литров.
  - Требуется: отмерить 6 литров.
  - Решение:
    - 1) отмерим 1 литр – наполним 9-ти литровый резервуар 5x2=10, останется 1 литр.
    - 2) добавим 1 литр к 5 литрам, получим 6 – 1 литр в пустой 9-ти литровый резервуар, и к нему добавим 5 литров.

Методы случайного поиска

- **Метод проб и ошибок**
  - пробы делаются в случайном направлении изменения значения оптимизируемой переменной с некоторым шагом.
    - Если при этом целевая функция приближается к своему экстремуму, то найденное решение запоминается, в обратном случае решение отвергается.
- **Локальный случайный поиск с возвратом**
  - Шаг 1. Осуществляется фиксированный шаг в случайно выбранном направлении.
    - При превышении новым значением целевой функции ***I(x1+∆)*** ее исходного значения ***I(x1)*** или в случае их равенства решение ***x1+∆*** бракуется и происходит возврат к исходному состоянию x1.
    - Если значение ***I(x1+∆)*** уменьшилось, то следующий шаг в случайном направлении делается из точки ***x1+∆*** .
  - Шаг 2. Осуществляется новый цикл поиска в соответствии с рекуррентным выражением: ***x(i+1)=x(i)+x(i+1)***

Метод динамического программирования

Общее правило построения алгоритмов:

- Найти такое разбиение задачи на две или более подзадач, чтобы оптимальное решение задачи содержало оптимальное решение всех подзадач, которые в нее входят.
- Написать рекуррентное соотношение.
- Вычислить оптимальное значение параметра для всей задачи.
  - **решение сверху вниз,** т.е. берем глобальную задачу, потом решаем только необходимые для нее подзадачи - задачу решается рекурсивно. Получив решение, процедура отмечает, что эта подзадача уже решена.
  - **решение снизу вверх – (**если же рекурсия невозможна) : решаются сначала элементарные подзадачи, потом только те, которые требуют результатов уже решенных подзадач и т. д., пока не будет решена общая задача.
- Если необходимо получить не только значение качества оптимального решения, но и найти само решение, то на шаге 2 нужно также запоминать некоторую дополнительную информацию о ходе решения решение каждой подзадачи. Этот шаг иногда еще называют **обратным ходом**.

Пример. Числа Фибоначчи

- Вычислить N чисел в последовательности Фибоначчи, — 1, 1, 2, 3, 5, 8, … — в которой первые два члена равны единице, а все остальные представляют собой сумму двух предыдущих. N меньше 100.
- Решение задачи: рекурсивная функция:

```
Function F(X:integer):longint;
   Begin
     if (X=1) or (X=2) then F:=1 else F := F(X-1) + F(X-2)
   end;
```

Пример. Числа Фибоначчи

- Создать массив, в котором хранятся значения функции:

```
 Var D : Array [1..100] of LongInt;
…
 Function F(X : integer) : LongInt;
    Begin
      if D[X] = 0 then
            if (X=1) or (X=2)
            then D[X] := 1
            else D[X] := F(x-1) + F(x-2);
      F := D[X]
    End;
```

Требования к задачам МДП

- **Оптимальность для подзадач**
  - задача **обладает свойством оптимальности для задач**, если оптимальное решение задачи содержит оптимальные решения её подзадач.
- **Перекрывающиеся подзадачи**
  - малость множества подзадач.
    - У оптимизационной задачи имеются **перекрывающиеся подзадачи**.
    - Алгоритмы, основанные на динамическом программировании, используют перекрытие подзадач следующим образом: каждая из подзадач решается только один раз, и ответ заносится в специальную таблицу; когда эта же подзадача встречается снова, программа не тратит время на её решение, а берёт готовый ответ из таблицы.

Жадные алгоритмы

- Жадный алгоритм строит решение посредством последовательности шагов, на каждом из которых получается частичное решение поставленной задачи, пока не будет получено полное решение.
- На каждом шаге выбор должен быть:
  - Допустимым, т.е. удовлетворять условиям задачи;
  - Локально оптимальным, т.е. наилучшим локальным выбором среди всех допустимых вариантов, доступных на каждом шаге;
  - Окончательным, т.е., будучи сделанным, он не может быть изменен последующими шагами алгоритма.
- Пример: набрать данную сумму денег минимальным числом монет последовательно выбирая монеты наибольшего возможного достоинства

Различие между жадными алгоритмами и динамическим программированием

- на каждом шаге **жадный алгоритм** берёт «самый жирный кусок», а потом уже пытается сделать наилучший выбор среди оставшихся, каковы бы они ни были;
- **алгоритм динамического программирования** принимает решение, просчитав заранее последствия для всех вариантов.