Оценка сложности вычислительных программ

лекция 22

План лекции

- Сложность программы по времени и по памяти
  - Основные понятия
  - Сложность в худшем случае, сложность в среднем
  - Оценка сложности для программы на языке Си
- Понятие оптимальной программы
  - Пример доказательства оптимальности
- Асимптотическая сложность и оптимальность

Основные параметры вычислений и данных

- Как число необходимых команд и ячеек памяти зависит от размера входных данных?

- Обозначим Time(A, x) и Space(A, x) число команд и ячеек памяти, необходимых программе A для обработки входных данных x

- Обозначим |x| >= 0 размер входных данных x

Примеры

- Умножение матриц MM
  - |x| = порядок матрицы x
  - Space(MM, x) = 3*|x|^2
  - Time(MM, x) = число умножений и сложений = 2 * |x|^3
- Сортировка массива простыми вставками I
  - |x| = длина массива x
  - Space(I, x) = |x|
  - |x| - 1 <= Time(I, x) = число сравнений <= |x| *(|x| - 1)/2
- Проверка на простоту пробными делениями TD
  - |x| = x
  - Space(TD, x) = |x|
  - 1 <= Time(TD, x) = число делений <= sqrt(|x|) - 1
  - Как изменится выражение для Time(TD, x), если взять |x| = число бит в записи x?

Минимальные требования к |.|

- Число команд, необходимых программе для обработки входных данных, должно стремиться к ∞ когда размер входных данных стремится к ∞
  - Time(A, x_k) -> ∞ при $|x_k|$ -> ∞

- Программа TD (проверка на простоту)
  - |x| = число битов в x

|x|	2	3	4	5	6	7
x	2-3	4-7	8-15	16-31	32-63	64-127
x*	3	5	13	31	59	127
Time(TD, x)	1	1	1-2	1-4	1-6	1-10

  - |x| = x

|x|	111	112	113	114	115	116	117
Time(TD, x)	2	1	9	1	4	1	2

Временная сложность

- Временной сложностью (сложностью по времени в худшем случае) программы A называется функция от размера входных данных $T(A, n) = \max\{ Time(A, x) \mid |x| = n \}$

Сложность по памяти

- Сложностью по памяти в худшем случае (пространственной сложностью) программы A называется функция от размера входных данных S(A, n) = max{ Space(A, x) | |x|=n }

Сложность в среднем 1/3

- Обозначим Input(n) = { x | |x| = n } множество входных данных размера n

- Обозначим P(n, x) вероятность входных данных x ∈ Input(n)
  - Можно считать P(n, x) = 1 / (число элементов в Input(n))
  - Иногда считают, что вероятность разных входных данных разная

- По определению вероятности $\Sigma_{x \in Input(n)} P(n, x) = 1$

Сложность в среднем 2/3

- Величина $\underline{T}(A, n) = \Sigma_{x \in Input(n)} Time(A, x)\, P(n, x)$ называется временной сложностью программы A в среднем

Сложность в среднем 2/3

- Величина $\underline{S}(A, n) = \Sigma_{x \in Input(n)} Space(A, x)\ P(n, x)$ называется сложностью по памяти программы A в среднем

Связь между разными мерами сложности

- Сложность в среднем не превосходит сложность в худшем случае

$$\underline{T}(A, n) <= T(A, n)$$
$$\underline{S}(A, n) <= S(A, n)$$

$$\underline{T}(A, n) = \sum_{x \in Input(n)} Time(A, x)\, P(n, x) <=$$
$$<= \sum_{x \in Input(n)} \max \{ Time(A, x) \mid |x| = n \}\, P(n, x) =$$
$$= T(A, n) \sum_{x \in Input(n)} P(n, x) = T(A, n)$$

- Сложность по памяти не превосходит сложность по времени

$$S(A, n) < = T(A, n)$$

  - В каждую ячейку памяти нужно хотя бы записать значение

Пример вычисления сложности по времени в среднем 1/3

- Возведение a в степень x методом повторных квадратов RS

```
RS(a, x):
   q = a
   u = 1
   for bit in запись x в двоичной с.с. :
      if bit == 1:
         u *= q
      q *= q
   return u
```

Пример вычисления сложности по времени в среднем 2/3

- Input(n) = { x | 2^{n-1} <= x < 2^n – 1 }
- |x| = число битов в x
- P(n, x) = 1 / (число элементов в Input(n)) = 1 / 2^{n-1}
- $\underline{T}$(RS, n) = (1 / 2^{n-1}) $\Sigma_{x \in Input(n)}$(|x| + (число битов = 1 в x) – 2) =

  = n – 2 + 1 + (1 / 2^{n-1}) $\Sigma_{x \in Input(n)}$ (число битов = 1 в x)

Пример вычисления сложности по времени в среднем 3/3

- Как известно, $k \cdot C(n, k) = n \cdot C(n - 1, k - 1)$

- $\Sigma_{x \in Input(n)}$(число битов = 1 в x) =
  $= \Sigma_{0<=k<=n-1} k \cdot C(n - 1, k) = \Sigma_{1<=k<=n-1} (n - 1) \cdot C(n - 2, k - 1) =$
  $= (n - 1) \cdot \Sigma_{0<=k<=n-2} C(n - 2, k) = (n - 1) \cdot 2^{n-2}$

- $T(RS, n) = n - 1 + (1 / 2^{n-1}) \cdot \Sigma_{x \in Input(n)}$(число битов = 1 в x) =

$$= n - 1 + (1 / 2^{n-1}) \cdot (n - 1) \cdot 2^{n-2} = 3 \cdot (n - 1) / 2$$

Оценка сложности с практической точки зрения

- Точное число команд и ячеек памяти на практике не важно
  - Зависит от набора команд
  - Для входных данных большого размера слагаемые низших составляют исчезающий % от общего числа команд и ячеек памяти
- Обычно приближенно оценивают сверху самое быстро растущее слагаемое в зависимости от размера данных
- Существуют разные методы построения приближенных оценок сверху для сложности программ

Как оценить сложность программы на языке Си?

- Обозначим T(A, n) оценку сверху для T(A, n) на основе записи A на языке Си
- T({ A1; A2; }, n) = T (A1, n) + T(A2, n)
- T({ if (C) A1; else A2; }, n) = T (C, n) + max(T(A1, n), T(A2, n))
- T({ for (i = N(n); i > 0; --i) A(i); }, n) = T (N(n), n) + $\Sigma_{0<i<=N(n)}$ T(A(i), n)
- T({ F(A1, A2, …, AN); }, n) = T(A1, n) + … + T(AN, n) + T(тело(F), n)
  - Не применимо, если F является рекурсивной

Оптимальные программы

- Программа A* называется оптимальной по времени в классе программ AA, если для любой программы A из AA и любого размера n входных данных T(A*, n) <= T(A, n)
- Для доказательства оптимальности программы по времени требуется оценка T(A, n) снизу
- Существуют разные методы построения приближенных оценок снизу для сложности программ

Дерево трасс исполнения

- Трасса исполнения программы для входных данных x – это множество пар вида (номер шага при обработке x, исполненная на этом шаге команда)

- Дерево трасс исполнения для входных данных размера n
  - Множество вершин = объединение трасс для всех входных данных размера n
  - Вершина (q, c1) является родителем вершины (r, c2), если q + 1 = r

- «Дерево трасс исполнения получается склеиванием общих префиксов»

Построение оценки снизу для поиска min и max -- 1/4

- Пусть AA – все программы для одновременного нахождения минимума и максимума в массиве

- Покажем, что сложность по числу сравнений оптимальной программы $3n/2 – 2$, и приведем оптимальную программу

Построение оценки снизу для поиска min и max -- 2/4

- Каждый шаг произвольной программы, решающей эту задачу, характеризуется 4 множествами элементов массива (A, B, C, D):
  - A = не участвовали в сравнениях
  - B = во всех сравнениях были больше
  - C = во всех сравнениях были меньше
  - D = в одних сравнениях были больше, а в других — меньше

Построение оценки снизу для поиска min и max -- 3/4

- Пусть $\lambda(a, b, c) = 3*a/2 + b + c - 2$
  - a, b и c -- число элементов в A, B и C
- Начинаем с $\lambda(n, 0, 0) = 3n/2-2$
- Заканчиваем $\lambda(0, 1, 1) = 0$

- При движении в дереве трасс исполнения от корня к самому глубокому листу λ уменьшается не более, чем на 1 – см. таблицу справа

- Следовательно, в худшем случае требуется не менее 3n/2 – 2 сравнений

Сравне-ние	(a, b, c, d)	Измене-ние λ
AA	(a − 2,b +1,c +1,d)	−1
AB	(a−1,b,c+1,d) | (a−1,b,c,d+1)	-1/2 | -3/2
AC	(a −1,b +1,c,d) | (a −1,b,c,d +1)	-1/2 | -3/2
AD	(a −1,b +1,c,d) | (a −1,b,c +1,d)	-1/2 |-1/2
BB	(a,b −1,c,d +1)	-1
BC	(a,b −1,c −1,d + 2) | (a,b,c,d)	-2|0
BD	(a,b −1,c,d +1) | (a,b,c,d)	-1|0
CC	(a,b,c −1,d +1)	-1
CD	(a,b,c −1,d +1) | (a,b,c,d)	-1|0
DD	(a,b,c,d)	0

Построение оценки снизу для поиска min и max -- 4/4

- Дан массив из n элементов x1, ..., xn
- Образуем пары x1, x2 ; x3, x4 ; …
- В каждой паре найдём минимум и максимум за одно сравнение
- Пусть m1, m2, … – массив минимальных элементов пар размера n/2
- Пусть M1, M2, ... – массив максимальных элементов пар размера n/2
- Минимальный элемент исходного массива среди mi
- Максимальный элемент исходного массива среди Mi
- Если на первом шаге был непарный элемент (n — нечётное), то на него потребуется ещё два сравнения с найденными минимумом и максимумом
- В итоге на каждую пару тратится 3 сравнения

Асимптотические оценки сложности

- Функции f и g называются функциями одного порядка, если найдутся такие c1 и c2, что для любого n

$$c1|g(n)| < |f(n)| < c2|g(n)|$$

- Обозначается $f \sim g$

- Функция f -- омега функции g, если найдется такая константа c, что $|f(n)| > c \mid g(n) \mid$ для всех n

- Обозначается $f(n) = \Omega(g(n))$

Асимптотически оптимальная программа

- Программа A* называется асимптотически оптимальной (оптимальной по порядку сложности) в классе AA, если $T(A^*, n) = \Omega(T(A, n))$ для любой другой программы A из AA

Асимптотически оптимальная программа

- Если A* и B* -- оптимальные программы в классе AA, то T(A*, n) = Ω(T(B*, n)) и T(B*, n) = Ω(T(A*, n)) и T(A*, n) ~ T (B*, n)

- Оптимальная асимптотическая сложность определена однозначно

Заключение

- Сложность программы по времени и по памяти
  - Основные понятия
  - Сложность в худшем случае, сложность в среднем
  - Оценка сложности для программы на языке Си
- Понятие оптимальной программы
  - Пример доказательства оптимальности
- Асимптотическая сложность и оптимальность

Классы сложности задач

- Под «задачей» будем понимать набор из трех объектов:
  - функция $P(.)$, которую требуется вычислить
  - функция измерения входных данных $|.|$
  - функция измерения числа операций $T(.,.)$ в алгоритме вычисления функции $P(.)$

Классы сложности задач

- Задача P не сложнее Q, если для любой программы QA, решающей задачу Q, найдётся программа PA, решающая задачу P, такая что T(PA, n) = O(T(QA, n))
- Обозначение $P \leq Q$
- Задачи P и Q, для которых одновременно верно $P \leq Q$ и $Q \leq P$, называются эквивалентными (по сложности)
- Обозначение $P >< Q$

Пример

- Рассмотрим следующие задачи:
  - M: умножение 2-х целых чисел a и b
  - D: деление целого a битовой длины $\leq 2m$ на целое b битовой длины m
  - S: возведение в квадрат целого a
  - R: обращение целого a
- Покажем, что $M >< D >< S >< R$

Пример

- Можно доказать, что для |x| = число битов в x сложность f(.) любого из этих алгоритмов
  - не убывает
  - f(m) >= m
  - af(m) <= f(am) <= a^2f(m) для a > 1

Пример

- M < S
  - ab = ((a+b)^2-a^2-b^2)/2
  - T(MA, m) = T(SA, m+1)+2T(SA,m)+O(m) = O(T(SA,m))
- S < R
  - a^2 = 1/(1/a-1/(a+1))-a
  - T(SA, m) = O(T(RA, c*m)) – так как делить нужно в c раз более точно

Пример

- R < M
  - x[i]=2*x[i-1]-a*x[i-1]^2
  - Сходится к 1/a и x[i-1]=1/a*(1- ε) ==> x[i]=1/a*(1- ε^2)
    - Почему?
  - T(RA, m) = O(T(MA,m))
- M >< S >< R
- D < M
  - a/b = a*(1/b)
- R < D -- очевидно