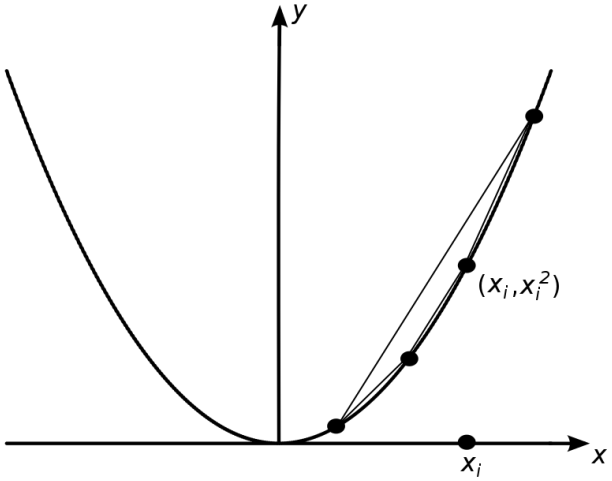
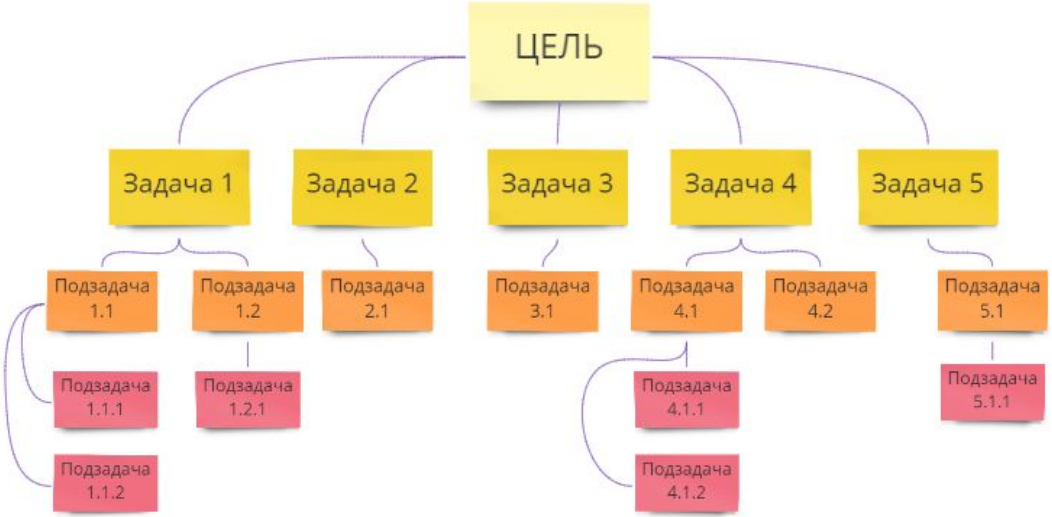


Метод декомпозиції

Обчислення опуклої оболонки методом декомпозиції

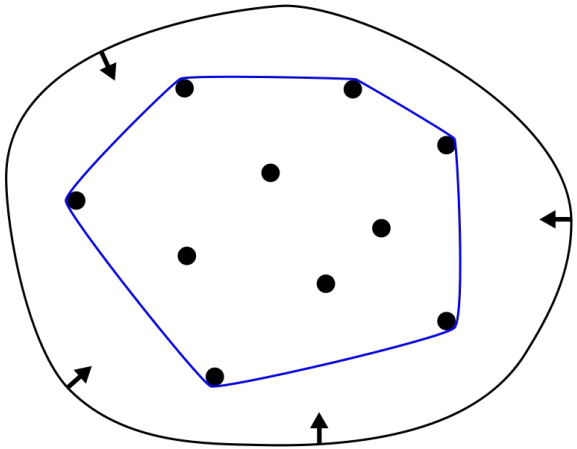
Метод декомпозиції

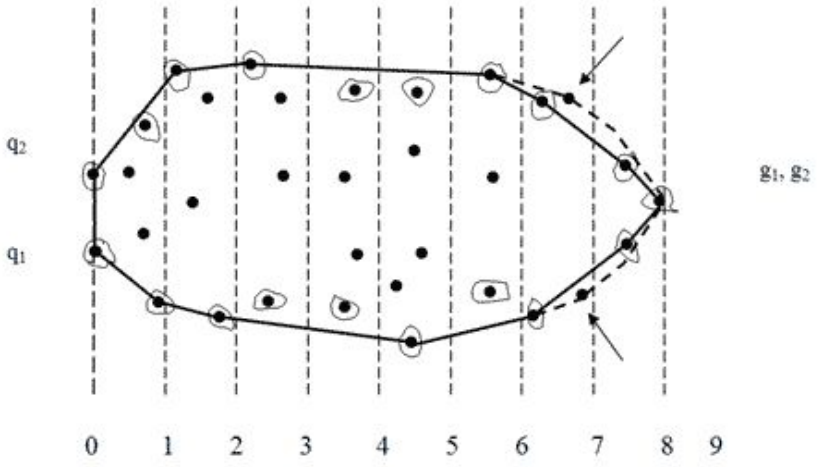
Метод декомпозиції - його ще називають розділяй і володарюй.
Можливо, є найважливішим і найбільш широко застосовним методом проектування ефективних алгоритмів. Він передбачає декомпозицію (розбиття) завдання на більш дрібні завдання, та на основі рішень цих дрібних завдань можна легко отримати рішення первісної задачі.



Опукла оболонка

Опуклою оболонкою множини називається найменша опукла множина, що містить. «Найменша множина» означає найменший елемент по відношенню до суперпозиції множин, тобто така опукла множина, що містить дану фігуру, що вона міститься в будь-якій іншій опуклій множині, що містить цю фігуру.



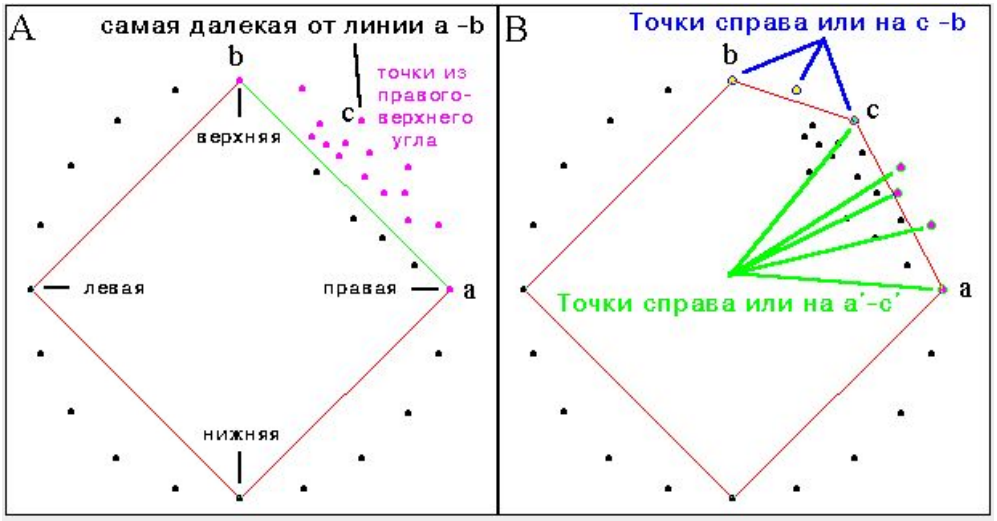


Алгоритми вирішення задачі

- Алгоритм швидкої опуклої оболонки
- Алгоритм Грехема
- Алгоритм Джарвіса
- Алгоритм Кіркпатріка
- Алгоритм Чана

Опишем алгоритм швидкої опуклої оболонки (QuickHull)

Цей алгоритм починає роботу зі створення чотирикутника, що сполучає крайні точки (дивись крок А). Тільки точки, що лежать поза ним можуть лежати на оболонці і будуть розглядатися в подальшому. Кожен лежить поза чотирикутника ділянку буде рекурсивно оброблений функцією Quickhull. На діаграмі нижче зображена обробка верхнього-правого кута.



Оцінка складності алгоритму

Складність алгоритму складається з складності побудови двох підмножин розглянутого множиниO(N) та складності вирішення підзадач для кожної з підмножин: T (N) = T (a) + T (b) + O (N)

У кращому випадку, коли задача розбивається на дві рівнопотужності підзадачі, складність алгоритму є рішенням рекурсивного рівняння:

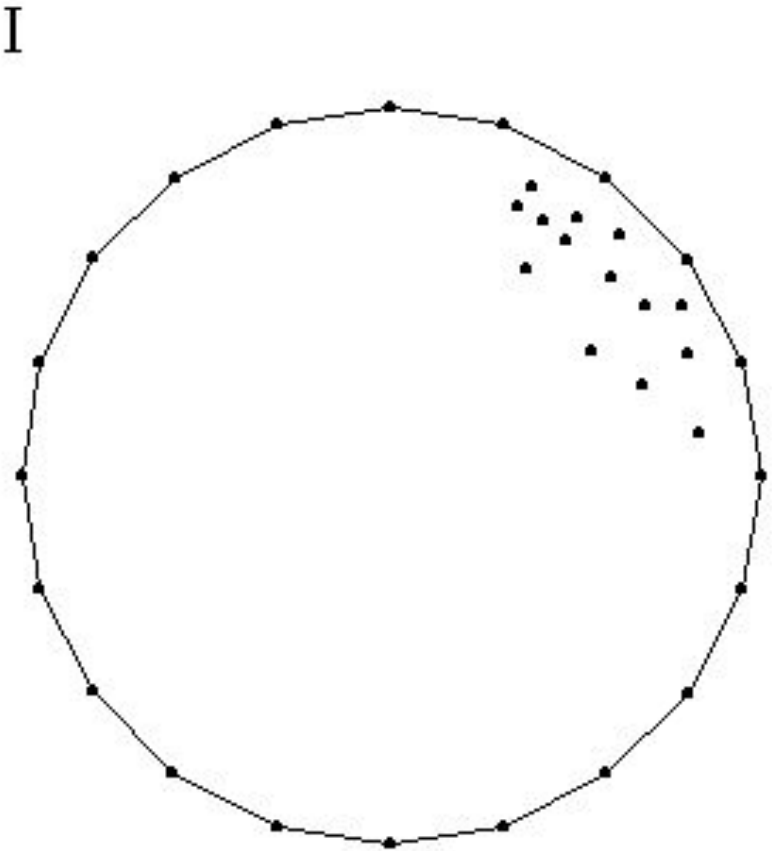
$T(N) = 2\ T(N/2) + O(N) \Rightarrow T(N) = O(N \log N)$

В гіршому випадку:

$T(N) = T(1) + T(N^1) + O(N) \Rightarrow T(N) = O(N^2)$

Гірший випадок

Найгіршу поведінку алгоритм демонструє в разі, якщо задані точки вже утворюють опуклий багатокутник, так як ніякі точки в цьому випадку не відкидаються. Тоді він робить N^2 операцій



```
void qh( int *pt, int n )
{
  /* DC step: select pivot point from pt */
  int pivotpos, pivot;
  int p1=pt[0], p2=pt[1];
  int *left1, *left2, leftcnt1, leftcnt2;
  /* DC step: select any pivot point from pt.
     We have p1==pt[0],p2==pt[1] */
  if (n==2) return;

  if (n==3) {
    /* one point (beyond old p1,p2) must belong to hull */
    belongs_to_hull[pt[2]] = 1;      /* saves a recursive call */
    return;
  }

  pivot = pivotize( pt, n );

  belongs_to_hull[pivot] = 1;
  leftcnt1 = n;

  left1 = delete_right( pt, &leftcnt1, p1, pivot );

  qh( left1, leftcnt1 );
  leftcnt2 = n;

  left2 = delete_right( pt, &leftcnt2, pivot, p2 );
  qh( left2, leftcnt2 );
}
```

Функція QuickHull

Повний код програми доступний на github:

https://github.com/LeraKulik/quickHull

Кулик В.М.