Тема 4. Быстрые деревья поиска

Лекция 14. Модификации B-деревьев

Определение. B+ деревом порядка n называется следующая модификация B-дерева.

Предполагается, что все элементы представляю собой пару (ключ, значение).

Концевые вершины дерева представляют собой упорядоченные массивы, содержащие пары (ключ, значение),и также все концевые вершины связаны в упорядоченный список.

Внутренние вершины содержет только ключи, при этом i-й ключ внутренний вершины есть первый ключ корневой вершины ее $(i+1)$-го подддерева. Т.е. все ключи $(i+1)$-го подддерева больше или равны i-го ключа родительской вершины, и соответственно все ключи i-го подддерева меньше этого i-го ключа родительской вершины.

также все концевые вершины лежат на одном уровне.

В каждой вершине от n до 2n элементов (кроме корня).

Таким образом, внутренние вершины B+ дерева представляют собой "надстройку" для поиска элементов концевых вершин, где также хранятся и "значения".

Определение. 2-3 дерево есть B-дерево порядка 1.

Таким образом, вершины 2-3 дерева могут иметь либо 1, либо 2 элемента и соответственно либо 2, либо 3 потомка.

Можно построить реализации таких деревьев по традиционной схеме, но здась мы рассмотрим еще одну фундаментальную возможность языка C++ — наследование.

Наследование

Понятие сложное и многогранние, поэтому пока только некоторые факты без исчерпывающей полноты.

Концепция родительский класс -> порожденный класс

Порожденный класс расширяет функциональность родительского (дополнительные члены, функции). Повторное использование кода, расширение на частные случаи.

Есть определенная дисциплина доступа к членам родительского класса из порожденного класса.

```
class A
{
   private:
     int x;
   public:
     int y;
   protected:
     int z;
};

 A a;
 B b;

 доступны внутри класса A
 a.x  a.y  a.z
```

```
class B : public A
{
    private:
       int xx;
    public:
       int yy;
};

    доступны внутри класса B
    b.y  b.z  b.xx  b.yy
```

```
   доступны снаружи
   a.y     b.y   b.yy
```

Переопределение функций

```
  class A                              class B : public A
  {                                    {
   public:                               public:
     int f() { return 1; }                 int f() { return 2;}
  };                                   };

   A a;
   B b;
   int k = a.f();                      int k = b.f();
\end{erbatim}
```

```
  Указатели и наследование, виртуальные функции

\begin{verbatim}
  class A
  {
     public:
        void f() { cout << " A::f "; }
        virtual void g() { cout << " virtual A::g "; }
     //   virtual void h() = 0;      // чисто виртуальная функция
  };

  class B : public A
  {
     public:
        void f() { cout << " B::f "; }
        void g() { cout << " virtual B::g "; }
        void h() { cout << " B::h "; };
  };

  A a;  B b;

  a.f();      " A::f "
  a.g();      " virtual A::g "
  // a.h();  невозможно!!!!

  b.f();      " B::f "
  b.g();      " virtual B::g "
  b.h();      " B::h "

  A *pa = &a;
  B *pb = &b;
```

```
pa->f(); pa->g(); pb->f(); pb->g();  то же самое

A *q = &b; // вот так !!!

q->f();   " A::f "
q->g();   " virtual B::g "
q->h();   " B::h "
```

Интерфейсы — класс с чисто виртуальными функциями дает заголовки функций, которые могут быть релизованы по-разному.

Преобразование типа

```
(type)x    double* a = (double*)24;
double* a = static_cast<double*>24;

const int * a;
int *b;
b = const_cast<int *>a;

static_cast<type>(x)   - допустимое преобразование на этапе компиляции
dynamic_cast<type>(x)  - допустимое преобразование в иерархии наследования
                         (при выполнении программы, может возникнуть отказ)
const_cast<type>(x)    - преобразование для снятия/назначения const
reinterpret_cast<type>(x)  - ‘‘произвольное’’ преобразовние
```

Пример: реализация 2-3 дерева через наследование.
базовый класс — узел бинарного дерева
порожденный класс добавляет одно значение и один указатель на потомка