

Paralelní a distribuované algoritmy

1MIT - 2007/2008

Implementace algoritmu Minimum extraction sort

Autor

Jiří Hrazdil, xhrazd06

Zadání

Implementujte řazení libovolného počtu hodnot pomocí algoritmu Minimum Extraction Sort v jazyce PM2 pro architekturu PRAM.

Popis algoritmu

Algoritmus pracuje nad binárním stromem, v jehož listech jsou na počátku uloženy řazené hodnoty. Všechny nelistové uzly obsahují procesor, který je schopen porovnat hodnoty potomků a menší z nich uložit do svého uzlu. Hodnota menšího z potomků se tedy posunuje stromem vzhůru, až dosáhne kořene – po $(\log n) + 1$ krocích. Následně do kořenového uzlu postupují další nejnižší hodnoty – vždy ve dvou krocích – nejprve se hodnota dostane do kořene, a pak je z ní odstraněna do pole s výsledkem. Jde o binární strom, tzn. počet listů je vždy mocninou dvojky. Abychom mohli řadit i hodnoty, jejichž počet není mocninou dvojky, musíme si u zbývajících listů poznamenat, že jsou prázdné. U každého uzlu si tedy pamatujeme hodnotu, kterou obsahuje, a také proměnnou, ve které uchováváme (ne)prázdnost uzlu.

Teoretická časová složitost

Nejmenší hodnota dosáhne kořene stromu po $(\log n) + 1$ kroku. Následně je zbývajících $n - 1$ hodnot ze stromu „odstraněno“ v pořadí od nejmenší vždy po dvou krocích – prvním je doručení hodnoty do kořene, druhým je odstranění hodnoty z kořene a uložení do výsledného pole.

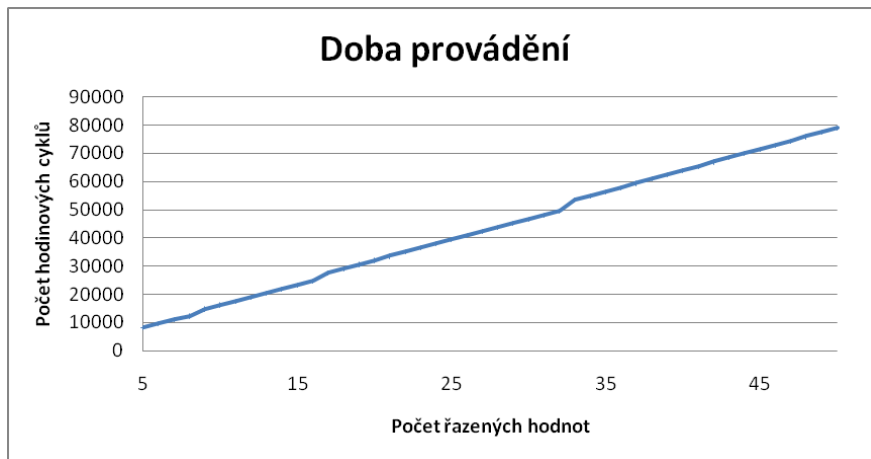
Celkový počet kroků je tedy $2 * (n - 1) + (\log n) + 1 = 2 * n + \log n - 1$. Dominantní složkou složitosti je n , jde tedy o algoritmus s lineární časovou složitostí.

Experimenty

K experimentování jsem použil skriptovací jazyk PHP a jednoduchý skript, který spustí řazení s náhodně vygenerovanými sadami hodnot s 5 až 50 prvky a z výstupu přečte počet spotřebovaných hodinových cyklů.

V následující tabulce jsou uvedeny hodnoty počtu hodinových cyklů pro různé počty řazených hodnot. V posledním sloupci je uveden počet cyklů na jednu řazenou hodnotu. Kolísání je způsobeno doplňováním počtu řazených hodnot na mocninu dvou. V grafu pod tabulkou vidíme lépe skutečný průběh časové složitosti – lineární, což odpovídá teoretické složitosti.

| počet hodnot | počet hodinových cyklů | poměr |
|--------------|------------------------|-------|
| 5 | 8253 | 1651 |
| 10 | 16284 | 1628 |
| 20 | 32101 | 1605 |
| 30 | 46661 | 1555 |
| 40 | 63904 | 1598 |
| 50 | 78924 | 1578 |



Komentovaný výpis algoritmu

```

program prl2;
sharedvar values : array of word; // pole razenych hodnot, sdilene vsemi procesory
        empty : array of boolean; // pole signalizujici neprazdnost uzlu
        result : array of word;
        n : word;
        leafs : word;
        kroku : word;
        resultIndex : word;
var i, j, i1, i2 : word;

procedure init;
var i : word;
begin
    read(leafs); // nacteni poctu vstupnich hodnot
    // ziskani nejblizsi vyssi mocniny 2 (neni-li hodnota promenne leafs mocninou dvou)
    n := 2;
    while (n < leafs) do
        n := n * 2;
    end;
    // v n je nyní neblizsi vyssi mocnina 2 od poctu razenych hodnot
    P := n - 1; // inicializace poctu procesoru - potrebujeme je pro nelistove uzly procesoru

    // vyprazdname pole empty
    for i := 1 to 2 * n - 1 do
        empty[i] := true;
    end;
    // nacteni hodnot do listu
    for i := n to n + leafs - 1 do
        empty[i] := false;
        read(values[i]);
    end;
    // vypocet poctu kroku k serazeni posloupnosti
    kroku := 2 * leafs + log(n) - 1;
    // nastaveni indexu pro zapis do pole vysledku
    resultIndex := 1;
end init;

procedure finish;
var i : word;
begin
    write(leafs); // vypis poctu razenych hodnot
    chwrite(32);
    for i := 1 to leafs do
        write(result[i]); // vypis serazenych hodnot
        chwrite(32);
    end;
end finish;

begin
    // proved vypocteny pocet kroku

```

```

for i := 1 to kroku do
  synchronize;
  // iteruj synchronne pres vsechny nelistove uzly (pres vsechny procesory)
  par j := 1 to P sync do
    // vypocet indexu potomku aktualniho uzlu v poli values
    i1 := 2 * j;
    i2 := i1 + 1;
    if (j = 1) and (not empty[j]) then
      // uzel je koren a neni prazdny
      // uloz vysledek
      result[resultIndex] := values[j];
      empty[j] := true;
      inc(resultIndex);
    elsif (empty[j]) then
      // aktualni uzel je prazdny
      // pokud jsou oba potomci neprazdni
      if ((not empty[i1]) and (not empty[i2])) then
        // a levý je mensi nez pravy
        if (values[i1] < values[i2]) then
          values[j] := values[i1];
          empty[j] := false;
          empty[i1] := true;
        else
          values[j] := values[i2];
          empty[j] := false;
          empty[i2] := true;
        end;
      elsif (empty[i1] and not empty[i2]) then
        values[j] := values[i2];
        empty[j] := false;
        empty[i2] := true;
      elsif (not empty[i1] and empty[i2]) then
        values[j] := values[i1];
        empty[j] := false;
        empty[i1] := true;
      end;
    end;
  end;
end;
end;
@CLOCK
end pr12.

```

Závěr

V projektu jsem implementoval algoritmus Minimum extraction sort a provedl s implementací praktické ověření shody časové složitosti teoreticky vypočtené a prakticky naměřené.