Structures de Tas et Files de priorité [hp03] - Exercices

Karine Zampieri, Stéphane Rivière

Unisciel algoprog Version 21 mai 2018

Table des matières

C++ - Tas et Files de priorité (Solution)

Mots-Clés Structures de tas ■
Requis Axiomatique objet, Modèles, Gestion des exceptions ■
Difficulté ●●○ (3 h) ■

Objectif
Cet exercice réalise des classes génériques de files de priorité de type `T`.

1 Fonctions de Tas

1.1 Opérations de tas

Écrivez les opérations de *Tas*.

Validez vos opérations avec la solution.

Solution C++ @[heapfuncs.cpp]

```
#ifndef HEAP_FUNCTIONS
#define HEAP_FUNCTIONS
#include <vector>
#include <functional>
using namespace std;

/**
  Insere un element dans un tas
  @param[in,out] v - un tas
  @param[in] last - indice de l'element a inserer
  @param[in] comp - fonction de comparaison
  Les elements v[0..last-2] forment un tas.
  Insere l'element v[last-1] et restaure la condition du tas.
*/
template<typename T, typename Compare>
void pushHeap(vector<T>& v, int last, Compare comp)
{
  // currentPos is an index that traverses path of parents.
  // target is value hlist[i] and is repositioned in path
  int currentPos = last-1;
  int parentPos = (currentPos-1)/2;
  T target = v[last-1];
  // traverse path of parents up to the root
  while (currentPos != 0)
  {
    // compare target and parent value
    if (comp(target,v[parentPos]))
    {
      // move data from parent position to current position
      v[currentPos] = v[parentPos];
      // update current position to parent position
      currentPos = parentPos;
      // compute next parent
      parentPos = (currentPos-1)/2;
    }
    else
      // heap condition is ok
      break;
  }
  // the correct location has been discovered. assign target
  v[currentPos] = target;
}

/**
  Fait descendre l'element v[first] dans le tas v[first..last(
```

```
  @param[in] v - un vecteur
  @param[in] first - indice bas de l'intervalle
  @param[in] last - indice haut (exclus) de l'intervalle
  @param[in] comp - fonction de comparaison
*/
template <typename T, typename Compare>
void adjustHeap(vector<T>& v, int first, int last, Compare comp)
{
  // start at first and filter target down the heap
  int currentPos = first;
  T target = v[first];
  // compute the left child index and begin a scan down
  // path of children, stopping at end of list (last)
  // or when we find a place for target
  int childPos = 2 * currentPos + 1;
  while (childPos < last)
  {
    // index of right child is childPos+1. compare the
    // two children. change childPos if
    // comp(v[childPos+1], v[childPos]) is true
    if ((childPos+1 < last) and comp(v[childPos+1], v[childPos]))
    {
      childPos = childPos + 1;
    }

    // compare selected child to target
    if (comp(v[childPos],target))
    {
      // comp(selected child, target) is true.
      // move selected child to the parent;
      // position of selected child is now vacated
      v[currentPos] = v[childPos];
      // update indices to continue the scan
      currentPos = childPos;
      childPos = 2 * currentPos + 1;
    }
    else
      // target belongs at currentPos
      break;
  }
  v[currentPos] = target;
}

/**
  Retire un element d'un tas
  @param[in,out] v - un tas
  @param[in] last - indice de l'element a supprimer
  @param[in] comp - fonction de comparaison
  Permute le premier et dernier element, supprime le dernier
  element puis restaure la condition du tas.
*/
template <typename T, typename Compare>
void popHeap(vector<T>& v, int last, Compare comp)
{
  // exchange the first and last element in the heap
  T temp = v[0];
  v[0] = v[last-1];
  v[last-1] = temp;
```

```
  // filter down the root over the range [0, last-1)
  adjustHeap(v, 0, last-1, comp);
}

/**
  Cree un tas
  @param[in,out] v - un vecteur
  @param[in] comp - fonction de comparaison
*/
template <typename T, typename Compare>
void makeHeap(vector<T>& v, Compare comp)
{
  // compute the size of the heap and the index
  // of the last parent
  int lastPos = v.size();
  int heapPos = (lastPos - 2)/2;
  // filter down every parent in order from last parent down to root
  while (heapPos >= 0)
  {
    adjustHeap(v, heapPos, lastPos, comp);
    --heapPos;
  }
}

#endif
```

1.2 Programme de test

Écrivez un programme de test.

Validez votre programme avec la solution.

Solution C++ @[pghpheap.cpp]

```
// Test les operations du tas
#include <iostream>
#include <vector>
#include <string>
using namespace std;
#include "heapfuncs.cpp"

/**
  Affiche un tas
  @param[in] hp - tas a afficher
  @param[in] txt - texte a afficher
*/
template<typename T>
void afficherHeap(const vector<T>& hp, const string& txt)
{
  cout << "==> " << txt << "[";
  for (int k = 0; k < hp.size(); ++k)
  {
    cout << hp[k] << " ";
  }
```

```
  cout << "]" << endl;
}

/** @test */
int main()
{
  int arr[] = {5, 9, 2, 7, 1, 3, 8};
  int arrSize = sizeof(arr)/sizeof(int);
  vector<int> vA, vB;

  // vA = maximum-heap et vB = minumum-heap
  for (int ix = 0; ix < arrSize; ++ix)
  {
    cout << "Insere " << arr[ix] << " dans les tas" << endl;
    vA.push_back(arr[ix]);
    pushHeap(vA, vA.size(), greater<int>());
    afficherHeap(vA, "A = ");
    vB.push_back(arr[ix]);
    pushHeap(vB, vB.size(), less<int>());
    afficherHeap(vB, "B = ");
  }

  // Vide les tas en affichant l'élément extremum
  cout << "Maximum-heap: ";
  while (not vA.empty())
  {
    popHeap(vA, vA.size(), greater<int>());
    cout << vA.back() << "  ";
    vA.pop_back();
  }
  cout << endl;

  cout << "Minimum-heap: ";
  while (not vB.empty())
  {
    popHeap(vB, vB.size(), less<int>());
    cout << vB.back() << "  ";
    vB.pop_back();
  }
  cout << endl;
}
```

2 FilePriorite

2.1 Classe FilePriorite

Écrivez un modèle de classes `FilePriorite<T>` qui modélise les files de priorité d'éléments de type `T`.

Validez vos méthodes avec la solution.

Solution C++ @[FilePriorite.cpp]

```
/**
  Constructeur par defaut
*/
template <typename T, typename Compare>
FilePriorite<T,Compare>::FilePriorite()
{}

/**
  Accesseur du nombre d'elements
  @return le nombre d'elements
*/
template <typename T, typename Compare>
int FilePriorite<T,Compare>::size() const
{
  return m_pq.size();
}

/**
  Predicat de conteneur vide
  @return Vrai si la PFile est vide
*/
template <typename T, typename Compare>
bool FilePriorite<T,Compare>::empty() const
{
  return m_pq.empty();
}

/**
  Insere un element dans la PFile
  @param[in] item - valeur de l'element
*/
template <typename T, typename Compare>
void FilePriorite<T,Compare>::push(const T& item)
{
  // insere item en fin de vecteur
  m_pq.push_back(item);
  // restaure la condition du tas
  pushHeap(m_pq, m_pq.size(), m_comp);
}

/**
  Supprime l'element de priorite maximale
  @pre la PFile n'est pas vide
```

```
  @throws une runtime_error exception si la PFile est vide
*/
template <typename T, typename Compare>
void FilePriorite<T,Compare>::pop()
{
  if (empty())
  {
    throw runtime_error("FilePriorite<T,Compare> pop(): file vide");
  }
  // met l'element en fin de vecteur
  popHeap(m_pq, m_pq.size(), m_comp);
  // supprime le dernier element du vecteur
  m_pq.pop_back();
}

/**
  constant version
  @return une const reference a element de priorite maximale
  @pre la PFile n'est pas vide
  @throws une runtime_error exception si la PFile est vide
*/
template <typename T, typename Compare>
const T& FilePriorite<T,Compare>::top() const
{
  if (empty())
  {
    throw runtime_error("FilePriorite<T,Compare> top(): file vide");
  }
  return m_pq[0];
}
```

Validez votre classe avec la solution.

Solution C++ @[FilePriorite.hpp]

```
#ifndef FILE_PRIORITE_CLASS
#define FILE_PRIORITE_CLASS
#include <vector>
#include <stdexcept>
using namespace std;
#include "heapfuncs.cpp"

/**
  File de priorite generique (= tas)
*/
template<typename T, typename Compare = greater<T> >
class FilePriorite
{
  public:
    FilePriorite();

    int size() const;
    bool empty() const;
    void push(const T& item);
    void pop();
    const T& top() const;
```

```
  private:
    vector<T> m_pq; //! file de priorite des elements
    Compare m_comp; //! fonction de comparaison
};
#include "FilePriorite.cpp"
#endif
```

2.2 Programme de test

Écrivez un programme de test.

Validez votre programme avec la solution.

Solution C++ @[pghppfile.cpp]

```
// Test de FilePriorite<T,Compare>
#include <iostream>
#include <cstdlib>
#include <functional>
using namespace std;
#include "Hasard.hpp"
#include "FilePriorite.hpp"

int main(int, char*[])
{
  cout << "Test de maximier de 'int'" << endl;
  FilePriorite<int, greater<int> > ctn;
  // Insere quelques elements dans le conteneur
  Hasard rnd;
  cout << "Insertion: ";
  for (int k = 0; k < 10; ++k)
  {
    int val = rnd.random(10);
    ctn.push(val);
    cout << val << " ";
  }
  cout << endl;
  // Extrait quelques elements du conteneur
  cout << "Extraction: ";
  for (int k = 0; k < 5; ++k)
  {
    cout << ctn.top() << " ";
    ctn.pop();
  }
  cout << endl;
  // Insere quelques autres elements dans le conteneur
  cout << "Insertion: ";
  for (int k = 0; k < 10; ++k)
  {
    int val = rnd.random(15);
    ctn.push(val);
    cout << val << " ";
  }
```

```
  cout << endl;
  // Vide le conteneur
  cout << "Vidage: ";
  while (not ctn.empty())
  {
    cout << ctn.top() << " ";
    ctn.pop();
  }
  cout << endl;
}
```

3 Références générales

Comprend [] ■